Comparison of explanations of simple and complex Machine Learning models

Ruiyu Wang

MSc in Advanced Computer Science,
School of Computing Science, University of Newcastle, U. K.
r.wang57@ncl.ac.uk

Abstract. Machine learning models make very accurate predictions, but their credibility is often questioned. Frequently, it is necessary to apply interpretive methods to the model's predictions in order to enhance their trustability. In this paper, two machine learning models (Decision Tree and Gradient Boosting Classifier) will be used to solve two classification problems. Their performances and SHAP(SHapley Additive exPlanations)interpretations will be compared to determine their level of trustworthy.
In order to validate the generated interpretations, in this paper, synthetic data generation techniques will be used to generate labels for classification problems according to custom rules and to verify that the model understands the custom rules.

Keywords: Machine Learning · SHAP · Synthetic data generation.

1 Introduction

With the success of machine learning (ML) in more and more areas in recent years, the need for explainability of ML has grown.

ML is now widely used in many domains, such as alphaGO and ChatGPT. However, ML models often fail to explain their predictions. For example, in alphaGO, the model will only give a method to win the game, but it will not explain why this method wins. In some specific domains, the loss of explanation is often accompanied by a loss of trust. There is also a loss of trust. For example, in medicine, if the model just gives a diagnosis without any explanation, it often fails to gain trust. This is because the cost of models that make incorrect predictions is often prohibitive.

Model interpretation in ML is the process of interpreting the behaviour and results of ML models [6]. A good explanation intuitively gives the reason why the model makes the predictions it does. This process can make the black box prediction process transparent.[2] The trustability and transparency of the model is ensured through the interpretation of the model[5][15], which strengthens the user's trust in the model's prediction results. At the same time, the interpretation of the model can often be used to improve the model or to gain some knowledge through the model.[2]

The complexity of generating explanations is directly proportional to the complexity of the ML algorithm that generates the model. Explaining the simplest linear regression can be done with simple arithmetic. Explaining complex deep neural networks is frequently, nevertheless, difficult to begin with. As a result, many model-agnostic interpretation methods have been proposed and widely used. SHAP (SHapley Additive exPlanations)[14] is one of the most representative methods.

At the same time, noise in datasets is one of the problems often faced in the field of ML. Datasets in nature are often not perfect, and noise can come from many sources, including incorrect collection by humans or instruments. Data with noise often negatively affects the prediction results of machine learning models [10][20].

The aim of this paper is to generate and compare interpretations of simple or complex ML models on synthetic datasets containing different levels of noise.

Section 2 gives a brief history of the development of machine learning interpretation techniques. Section 3 describes the experimental setup and datasets generation method. Section 4 will describe some of the methods used in the experiments, such as HalvingGridSearchCV and nested cross-validation. Section 5 and Section 6 describe the experimental procedure and analyse the results. Finally, Section 7 summarises the results of the experiments and gives conclusions about the comparison experiments.

2 Background of Interpretation

Interpretation methods for ML models can usually be divided into two types: model-specific or model-agnostic interpretation methods[6].

2.1 Model-specific Interpretation

For simple models (e.g. linear regression, logistic regression, decision trees, etc.), assumptions about the distribution are often made in advance[6]. These assumptions lead to a simpler structure of the models, making them easier to interpret. For example, Before using linear regression, the user needs to first assume that the predictions are linear. Linear regression generates predictions by linearly combining the features. Gauss has been using linear regression since the nineteenth century[8],this is precisely because the principle of linear regression is very simple: find a line in space such that the sum of the distances from each data point to the line is the shortest.

However, model-specific interpretation methods tend to apply only to simple models. The internals of highly complex models are not analysed and interpreted.

2.2 Model-agnostic Interpretation

Many model agnostic approaches have been proposed in recent years. These methods involve interpreting the model after training. Among them, the most

representative are LIME and SHAP. SHAP is used in this paper because it is an improvement of LIME with better functionality[4].

LIME In 2016, LIME was proposed by Ribeiro et al[18]. LIME is interpreted by inputting individual data points into the model to capture individual predictions. Perturbation data is then input to capture changes in the predictions to generate an interpretation of the model.

SHAP Lundberg and Lee presented SHAP in 2017[14]. SHAP is based on the Shapley value in game theory [21], a method of allocating profits according to contributions. SHAP can decompose a prediction into the contributions made to the prediction by each feature of the input data. While highlighting the contribution of individual features, SHAP also indicates the impact of interactions between features on the prediction [14]. In other words, the biggest difference between SHAP and LIME is that SHAP introduces the Shapley value to calculate the contribution made by features to the prediction[14].

Another key difference between LIME and SHAP is that SHAP can provide global interpretations of the entire dataset[14], while LIME tends to provide localized interpretations of an instance of data[18]. Furthermore, SHAP offers comprehensive visualization tools to display the interpretations as a graph.

From 2017 to the present, SHAP has undergone many updates, including the launch of TreeSHAP in 2019[13]. This is the SHAP variant for tree-based machine learning models such as decision trees, random forests and gradient boosting trees. Compared to the original KernelSHAP, TreeSHAP is faster, but somewhat more model-specific.

3 Experiment Preparation

3.1 Experimental Design

This experiment can be divided into three parts: generating synthetic datasets (with different levels of noise), training the model using machine learning algorithms, and generating model interpretations using SHAP.

To confirm the trustability of the generated interpretations, it is necessary to generate synthetic datasets. This is because the original dataset lacks knowledge of the relationship between labels and attributes. It is not possible to ascertain whether the explanation finds a solution to the problem or not by using the attributes and labels from the original dataset directly. This project will generate new labels from the attributes based on custom rules, and then use the attribute data with the new labels as a training set. Through this process, it is possible to analyse whether the model has found a customised rule or whether it has used other rules for prediction.

Noise can be divided into class noise and attribute noise [23]. Class noise is the misclassification or absence of labels. Attribute noise, on the other hand,

are outliers of input attributes. As the SHAP method calculates the contribution of each attribute to the prediction. In this project, attribute noise will be added to the dataset during the synthetic data generation stage. To add noise to the dataset, a rule is first artificially created to predict labels based on the attribute data. The special attributes are associated with the rule while the normal attributes are not. Two methods can be used to add noise when creating a synthetic dataset: (1) Adding the same level of noise to each attributes (2) Add different noise level to special attributes and the other.

Overall, custom rules for label generation enable defining a training objective for a machine learning model, which is to discover custom rules. If the interpretation shows that the model identified the rule, it denotes that the model might be reliable. However, if the interpretation indicates that the model failed to identify the rule, it suggests that its prediction process might not be reliable. Introducing noise enables assessing whether the model can still identify rules for obtaining labels from attributes despite subtle changes, hence providing a comparison.

3.2 Datasets Selection

Several machine learning benchmark datasets will be used as the original datasets in the experiments. The synthetic dataset necessary for training is generated from original datasets by defining rules to create new labels and adding noise.

The mushroom classification dataset[1] from the UC Irvine(UCI) Machine Learning Repository and the Stellar classification dataset[7] from the Kaggle competition were selected.

Mushroom Classification Dataset The dataset contains 8124 samples and 22 variables (odor, shape, color, etc. of different parts of the mushroom)[1]. Predict whether a mushroom is poisonous or not based on various attributes. It is important to note that the attribute values in this dataset are all categorical attributes. This means that these attributes have a fixed range of values and noise is added according to the range of values of the attributes.

Stellar Classification Dataset The dataset consists of data from space observations made by the Sloan Digital Sky Survey (SDSS)[3]. The dataset includes information about the instruments involved in the observations and spectral information about the stars observed. The dataset classifies stars into three categories, galaxies, quasars, and stars, and contains 17 classification attributes and a column of classification labels.

3.3 Noise Addition Method

Mushroom Classification Dataset Since the attributes in the mushroom categorical dataset are all categorical attributes, noise must be generated according to the range of values of each attribute in order not to change the nature of the dataset. Adding noise to the mushroom dataset uses the method of generating

random noise based on the range of values of the attributes and replacing the values in the original dataset. In this article, when noise is added to a dataset, it is added based on attributes. The method for adding noise to one attribute in mushroom dataset is as follows:

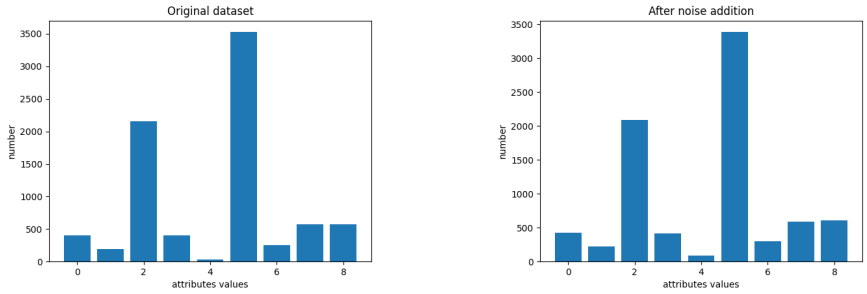
First, the column to which the noise is to be added is identified by passing external parameters.

Then, the global distribution of distinct values is extracted from that column of the original dataset.

Then, based on the global distribution and the noise ratio to be added, the number of distinct noise values to be generated is calculated.

Finally, based on the number of noise values, the number of indexes corresponding to each value is extracted from the original dataset and the data pointed to by the indexes is changed to other values.

Fig 1 shows the change in the number of different values of the odor attribute in the mushroom classification dataset before and after the addition of 5 percents noise. Sub-fig. 1(a) demonstrates the number of different odors before adding noise, while sub-fig. 1(b) displays the number of odors after adding five percent of noise.



(a) Original dataset (b) After noise addition

Fig. 1. Comparison before and after adding noise

The proportion of each odor before and after adding noise remained almost unchanged, as shown in Fig. 1, although there were some minor changes in the number of each odor.

Stellar Classification Dataset In a stellar categorical dataset, all data is numeric, which means that all attributes in the dataset are continuous attributes. Adding noise to continuous attributes is much simpler than adding noise to categorical attributes, where noise can be generated based on various mathematical distributions and added directly to the dataset. Common types of noise include Gaussian noise, Perlin noise, and others.

Gasussian Noise Gaussian noise is a frequently-occurring form of random signal noise, which is also referred to as normally distributed noise, or white noise[11]. This type of noise is the result of adding multiple independent random variables together, which follow a Gaussian distribution. This distribution is also known as a normal distribution. Gaussian noise is ubiquitous in numerous natural and engineering systems, such as electronic devices, communication systems, image processing, and measurement equipment.

Gaussian distribution has the following characteristics:

Mean: Gaussian noise has a mean of 0, which means it has an expected value of 0.

Variance: the variance determines how much the Gaussian noise spreads around the mean. A smaller variance will concentrate the noise near the mean, while a larger variance will result in a wider range of noise.

Symmetry: the Gaussian distribution is symmetrical, with a peak at the mean and a gradual decay to the sides.

The mathematical expression for Gaussian noise is

$$f(X) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-} \frac{(x-\mu)^2}{2\sigma^2} \tag{1}$$

where f(x) is the probability density function, μ is the mean, and σ is the standard deviation. One can vary the standard deviation of the Gaussian distribution in practice to achieve various levels of Gaussian noise.

Because Gaussian noise follows a Gaussian distribution, most noise values are clustered near the mean, and noise values far from the mean are less likely to occur. This makes Gaussian noise very useful statistically

Perlin Noise Perlin noise is an algorithm created by Ken Perlin in 1985 to generate continuous, smooth, pseudo-random noise[17]. It is commonly used in computer graphics, animation, and game development to create natural, realistic-feeling textures, terrain, clouds, water waves, and other effects[9].

Perlin noise is known for its ability to generate continuous noise values with smooth transitions in space, without the presence of noticeable edges or abrupt changes. This property makes it exceptionally useful in simulating natural environments and generating textures[17].

Perlin noise generation is achieved through interpolation. This method combines multiple random gradient vectors with the input point positions to obtain a continuous noise value[9].

The Perlin noise algorithm can be broken down into the following steps:

Create a grid for the input points and determine the stochastic gradient vector for each grid point. Map the input points to the grid and calculate the offset from each input point to the nearest grid point. At each grid point, the noise value is calculated by using the offset and the stochastic gradient vector. The noise values at various grid points are interpolated to produce the final Perlin noise value.[9]

There are five parameters in the Python noise library that can be used to control the different Perlin noises generated:

Scale: The scale is a factor used to control the coarseness of the noise texture. Lower values will generate finer textures, and higher values will produce coarser ones.

Octaves: Octaves refer to the number of frequencies used to superimpose the noise textures. Adding more octaves increases detail and complexity, but could also raise the computational demand.

Persistence: Persistence controls how much each frequency contributes to the noise texture. The values range from 0 to 1, where higher values indicate a stronger effect of high-frequency noise on the final outcome.

Lacunarity: The separation parameter, is used to regulate the distance between successive frequencies. It impacts the extent of each frequency.

Seed: The seed value determines the beginning of the noise generation process. Using the same seed value will result in the same noise pattern, facilitating production of a specific noise texture.

4 Experiment

This paper selected both decision tree and Gradient Boosting tree Classifier(GBC) in the experiments. HalvingGridSearchCV (HSGCV)[12] was used to parameterize the model, and the process of HSGCV was evaluated using nested cross-validation[22] in the experiments.

4.1 Preprocessing

In the pre-processing of the data before training, the following treatments were applied:

Extract the 'class' attribute from both datasets as the label set.

The attribute named 'veil-type' has been removed from the mushroom classification dataset because it contains only one value and was considered to have no impact on the prediction.

The attributes in the dataset for stellar classification are restricted to redshift and the five filters available in the photometric system. This is due to their insignificance in generating classification predictions, as these attributes comprise of various IDs, times, etc.

The stellar dataset is a triple classification problem with three labels: 'star', 'galaxy', and 'quasar'. The dataset is transformed into a binary classification problem by combining the 'galaxy' and 'quasar'. There were two reasons for this transformation: to address some degree of class imbalance and to allow TreeSHAP to generate interpretable explanations for the GradientBoostingClassifier(GBC) because TreeSHAP can not generate explaination while GBC is handling a muti-class classification problem. After combing, dataset has two class: 'star' and 'other'.

4.2 Custom Rules

Mushroom dataset the custom rules are:

If the Mushroom 'odor' attribute is not none and the 'population' attribute is not 'SEVERAL' or 'SOLITARY', the mushroom is 'poisonous'.

If the mushroom has a brightly 'cap-color' and 'gill-color', the mushroom is 'poisonous'.

The mushroom is 'poisonous' if its 'habitat property' is 'WASTE'.

In all other cases the mushroom is 'edible'.

The above attribute columns are special columns in the Mushroom Classification dataset. This leaves about 51% of mushrooms classified as 'poisonous'.

Stellar dataset The rule relate to formula in astrophysics[19]. However, due to the intricacy of the accurate calculations, the rules only create an approximation:

Calculate the mean of data on five filters.

While 0<'Redshift'<1, use 'Redshift' times the speed of light divided by the Hubble constant to get the distance.

Calculate of absolute magnitudes using formula:

$$M = m + 5 - lg(d) \tag{2}$$

Where M means absolute magnitudes, m means mean, d means distance.

If M $\in$ (-10,17), the stellar is classified as 'star'

In other case, stellar is classified as 'other' Since the five filter data work together, here, only the 'redshift' is set as a special column.

After applying this rule on dataset, about 40% instances were classified as 'star'.

4.3 HalvingGridSearchCV

Optimizing hyperparameters (fine-tuning) is a crucial component of machine learning that enables finding the hyperparameters leading to optimal model training outcomes.

However, manually adjusting the parameters can be tedious. Two automatic hyperparameter tuning methods are widely used in the model generation process, namely GridSearchCV and RandomizedSearchCV. GridSearchCV requires a hyperparameter grid, provided by humans. Then, the algorithm exhaustively combines the parameters in the grid to find the combination that provides the best training results. In contrast, RandomizedSearchCV requires artificially defining a range of values for the parameters and finding the hyperparameter combination that produces optimal training results within the specified parameter range.

HSGCV can be seen as an enhanced version of GridSearchCV. Employing GridSearchCV can be a very time-consuming task. HSGCV speeds up the hyperparameter optimisation process through a technique called successive halving[12]. Successive halving refers to the use of multiple iterations in the hyperparameter search process. Each iteration involves selecting a subset of the parameters for training, and then eliminating less effective parameters to gradually converge towards the optimal ones.

The advanced performance and ability of HSGCV to significantly reduce the time required for hyperparameter optimization made it suitable as an experimental function in Scikit-learn[16], a very popular machine learning library in Python.

4.4 Nested Cross-validation

Hyperparameter optimization often requires cross-validation(CV). K-fold cross-validation is a common technique used in CV, where the dataset is partitioned into K segments. During each iteration, one segment is used as validation data, while the remaining segments are used as training data. After each iteration, the average training results are computed, which provides an estimate of the model's generalization ability.

Nested cross-validation[22] is a technique that involves two layers of cross-validation, where the selection of hyperparameters occurs within the inner layer of cross-validation, which is then followed by evaluating the performance in the outer layer of cross-validation.

A thorough evaluation of the hyperparameter search process was conducted through nested cross-validation. We ensured that the hyperparameter search process identifies the best combination of hyperparameters to obtain the optimal model.

4.5 Experiment details

Fold of CV Due to the relatively small size of the mushroom classification dataset (8124 instances), a higher number of folds was used for cross-validation: fold equals 10 for both the inner CV and outer CV. In contrast, the stellar classification dataset contains a large number of samples (100,000 instances), fewer cross-validation folds were used: To expedite the training process,a 5-fold for both the inner CV and outer CV.

Noise level setting For mushroom dataset, 4 noise levels are set. For each of the three experiments, the lower level of noise was set to five of 0 %, 5%, 10%, and 20%, and the higher level of noise was the lower level of noise plus 5%.

For Stellar dataset, 4 noise levels are set, which are no noise, simple noise, medium noise, and complex noise.

5 Experiment Result

After training, the scores generated by nested cross-validation are available with SHAP-generated interpretations. Nested cross-validation in this paper uses the F1_macro score as an evaluation metric, presented in a table with data formatted as mean +/- standard deviation. SHAP's bar plot interpretation, beeswarm plot interpretation and heatmap plot are used in this paper.

The Bar plot shows the feature importance analysis derived from the SHAP interpretation.The Beeswarm plot shows the effect of attribute value on the prediction results ranked according to the feature importance.The Heatmap plot clusters the interpretations of each sample according to the similarity, which is used to show the global interpretation of the model on the whole dataset.

Since it takes a very long time to generate the heat map interpretation for the whole dataset, in this paper, the heat map interpretation is generated by randomly sampling the complete dataset (random sampling is done by disrupting the dataset and taking the first N) to generate the heat map interpretation for N samples.

5.1 Mushroom dataset

Same noise level for all attributes The decision tree and GBC were trained separately on the mushroom classification dataset, which contained different levels. As a result, the following outcomes were obtained:

Table 1 shows the performance of the two models on the mushroom classification datasets with different noise level(same level on each attribute)

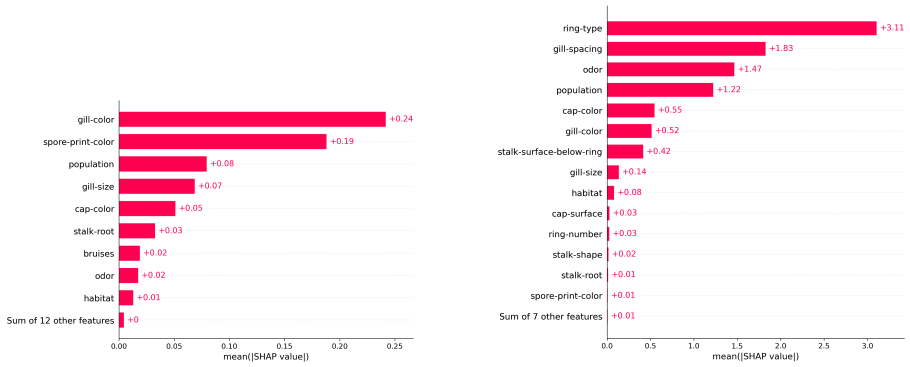
Table 1. F1-score from CV for same noise level on mushroom dataset

Noise level	Result from DT	Result from GBC
0%	1.000 +/- 0.000	1.000 +/- 0.001
5%	0.985 +/- 0.004	0.992 +/- 0.004
10%	0.969 +/- 0.010	0.989 +/- 0.004
20%	0.962 +/- 0.007	0.977 +/- 0.007

As can be seen in Table 1, as the noise level rises, the F1-score of both models produces minor reductions. Among them, the decision tree received a greater impact compared to the GBC.

Noise level 0% Both models achieved a 100% F1-score on the noise-free dataset as shown in Table 1. The confirmation of whether the two models identified the defined rules can only be made through further analysis of SHAP's interpretation.

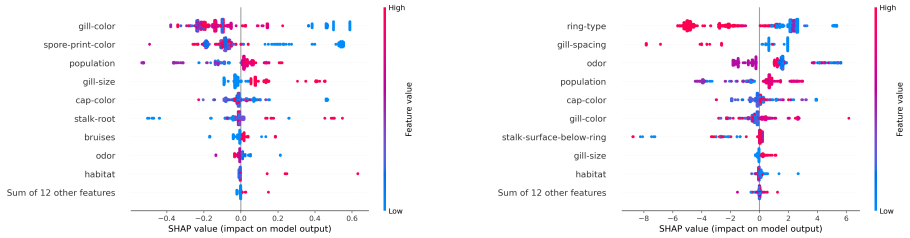
Fig. 2 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 2. SHAP bar plot for same noise level 0% on mushroom dataset

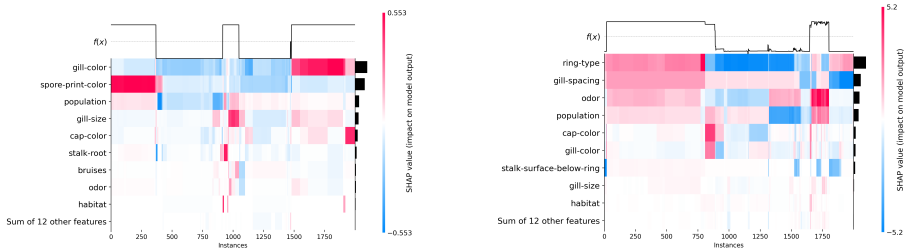
Fig. 3 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 3. SHAP beeswarm plot for same noise level 0% on mushroom dataset

Fig. 4 shows the heatmap plots generated by SHAP.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 4. SHAP heatmap plot for same noise level 0% on mushroom dataset

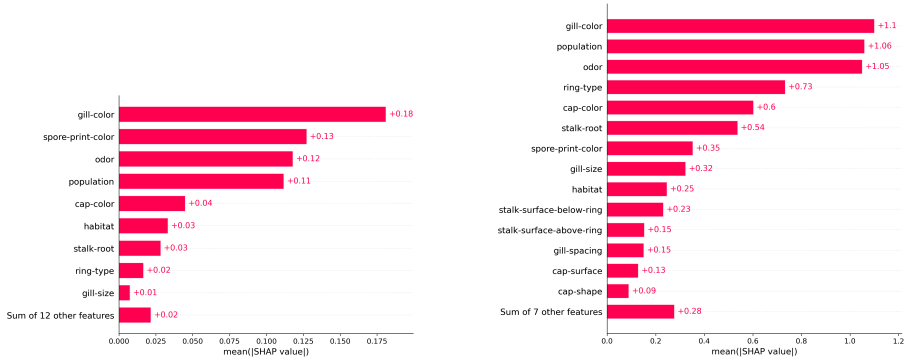
As can be seen in Fig. 2, the importance of the features produced by the decision tree model is very different from the importance of the features produced by the GBC. Odor was not considered an important attribute in the decision tree, which may have resulted in the model failing to learn the custom rules. On the other hand, the GBC has assigned importance to the two irrelevant attributes and has included them in all the rules. It is possible that the two irrelevant attributes were introduced due to the correlation between the attributes.

While working on the mushroom dataset (refer to Fig. 3), it is difficult to specify attributes based on high or low attribute values due to the categorical nature of the attributes. However, the beeswarm plot produced by GBC indicates a certain adherence to the rule where the presence of a specific 'odor' value has a negative effect on the classification as poisonous, while a specific 'habitat' value has a positive effect on the classification as poisonous. This is comparatively more difficult to observe while analyzing the beeswarm plot produced by decision trees.

Similarly, in Fig. 4, when the two rule-independent attributes are removed, the GBC produces a heatmap plot that fits the rule very well: there are three scenarios when categorised as poisonous (1) 'odor' and 'population' act together. (2) 'cap-color' and 'gill-color' act together. (3) The 'habitat' effect. This cannot be observed on the heatmap plot generated by the decision tree model.

Overall, although the two models get the same F1-score(100%), the GBC generates explanations that fit the rule better, so the GBC model's predictions should be more trustworthy.

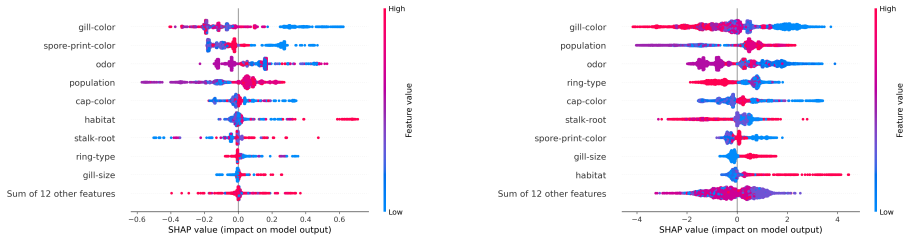
Noise level 5% Table 1 shows a very slight decrease in the F1 score for both models after adding 5% noise to the dataset. Fig. 5 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 5. SHAP bar plot for same noise level 5% on mushroom dataset

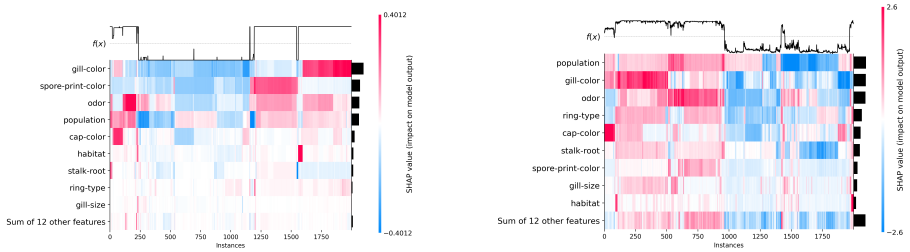
Fig. 6 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 6. SHAP beeswarm plot for same noise level 5% on mushroom dataset

Fig. 7 shows the heatmap plots generated by SHAP from these two models.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 7. SHAP heatmap plot for same noise level 5% on mushroom dataset

As can be seen in Fig. 5, after adding 5% noise, both models use more features in their prediction strategies. Both models identified strategy-related attributes as important.

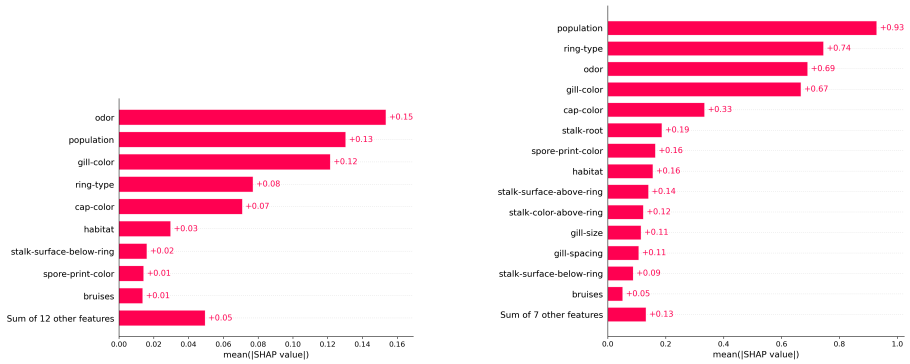
However, in Fig. 6, the interpretation becomes complicated with the addition of noise compared to the interpretation in the absence of noise.

In Fig. 7, it can be seen that although the interpretation becomes complex, it still roughly conforms to the custom rules.

Overall, both models' interpretations fit the custom rule well when there is 5% noise, but GBC has a higher F1-score.

Noise level 10% As can be seen in Table 1, the F1-score of both models further decreased when the proportion of noise in the dataset increased to 10%.

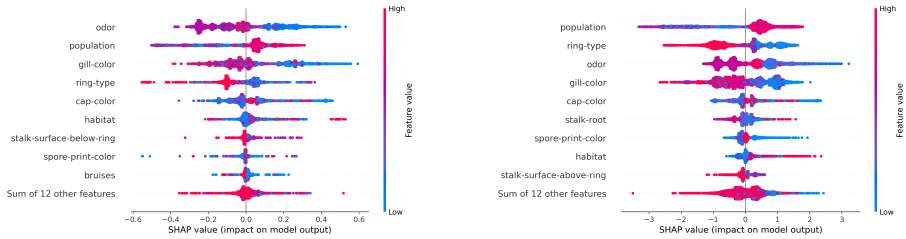
Fig. 8 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 8. SHAP bar plot for same noise level 10% on mushroom dataset

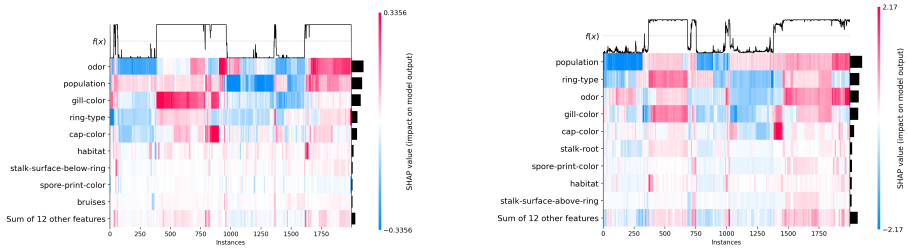
Fig. 9 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 9. SHAP beeswarm plot for same noise level 10% on mushroom dataset

Fig. 10 shows the heatmap plots generated by SHAP from these two models.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 10. SHAP heatmap plot for same noise level 10% on mushroom dataset

In Fig. 8, it is shown that as the noise increases to 10%, both models consistently consider several attributes associated with the rule to be the more important ones. In addition, both models consider 'ring-type' as a more important attribute. This should be due to the fact that 'ring-type' is highly correlated with rule-related attributes in the original dataset.

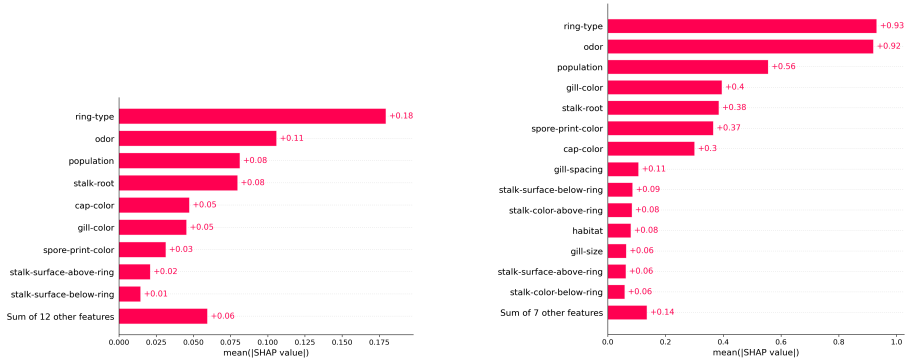
In Fig. 9, it is shown that as the noise increases from 5% to 10%, the areas of regular correlation on the beeswarm map become much more complex, with very cluttered colors.

In Fig. 10, it can be seen that an increase in the noise level complicates the interpretation of the model. Although both models still fit the rule roughly.

Overall, both models still roughly fit the rules, but the decision tree has more misclassification than the GBC.

Noise level 20% As can be seen in Table 1, when the noise rises to 20%, both models show a more significant decrease in F1-score.

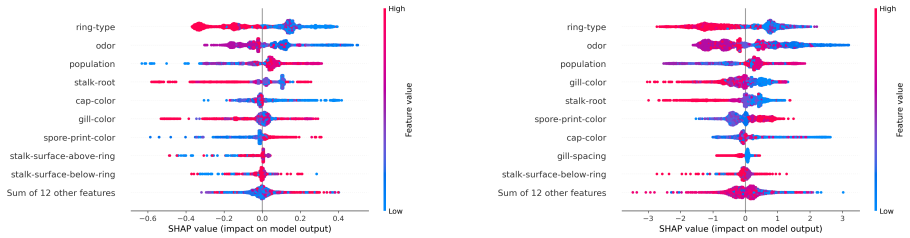
Fig. 11 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 11. SHAP bar plot for same noise level 20% on mushroom dataset

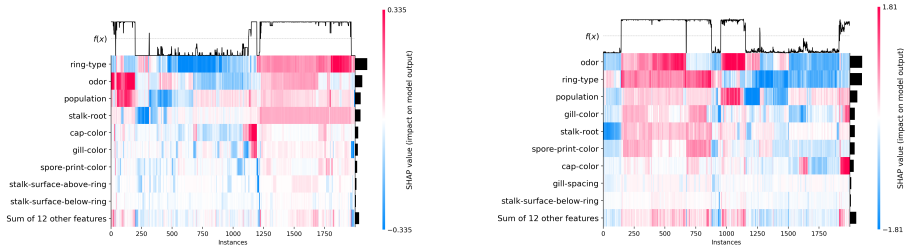
Fig. 12 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 12. SHAP beeswarm plot for same noise level 20% on mushroom dataset

Fig. 13 shows the heatmap plots generated by SHAP from these two models.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 13. SHAP heatmap plot for same noise level 20% on mushroom dataset

As shown in Fig 11, when the noise rises to 20%, both models no longer use 'habitat' as an important attribute, instead introducing some attributes outside the rules.

At the same time, as shown in Fig. 12, the beeswarm plots become very complex.

Fig. 13 shows that there is a big difference between prediction strategies and custom rules. This is evidenced by the fact that many rule-related attributes(e.g. 'habitat','cap-color') can no longer have an impact on predictions.

All in all, while noise level equals 20%, neither model can be trusted well.

Special attributes have different noise level The decision tree and GBC were trained separately on the mushroom classification dataset, which the noise level on special attributes is 5% higher or less than noise level on the other attributes. As a result, the following outcomes were obtained:

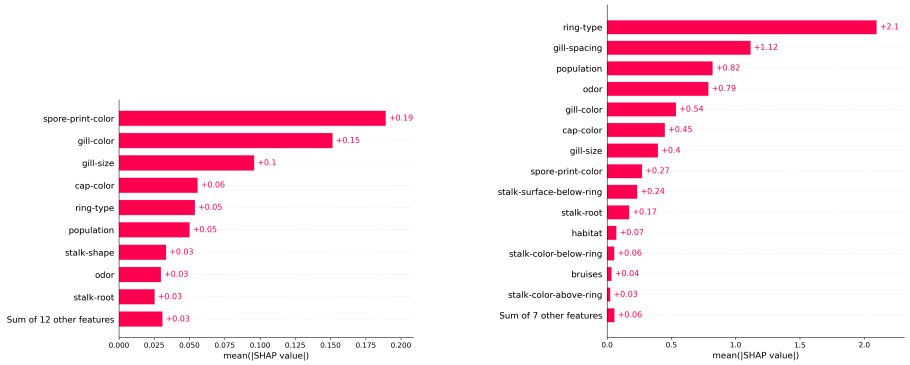
Table 2 shows the performance of the two models on the mushroom classification datasets with different noise level(special attributes have higher noise level)

Table 2. Mean F1-score from CV for same noise level on mushroom dataset

Normal Column Noise level	Special Column Noise level	Result from DT	Result from GBC
0%	5%	0.990 +/- 0.003	0.992 +/- 0.002
5%	10%	0.969 +/- 0.006	0.978 +/- 0.004
0%	5%	0.997 +/- 0.003	1.000 +/- 0.000
5%	10%	0.978 +/- 0.006	0.988 +/- 0.004

As can be seen in Table **??**, as the noise level rises, the F1-score of both models produces a certain decrease. Among them, the decision tree received a greater impact compared to the GBC. Comparing with Table 1, the F1-score decreased more in Table 2.

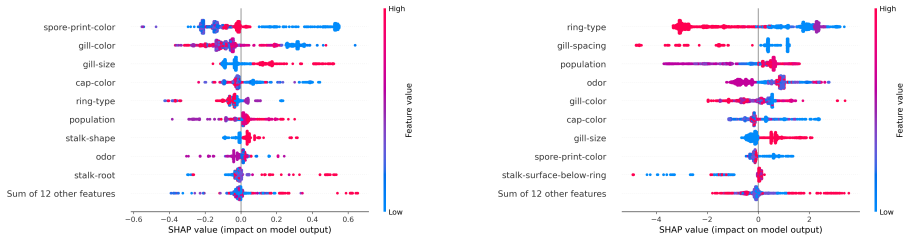
Special columns 5% others 0% Fig. 14 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 14. SHAP bar plot for special attributes have 5% noise and others have 0%

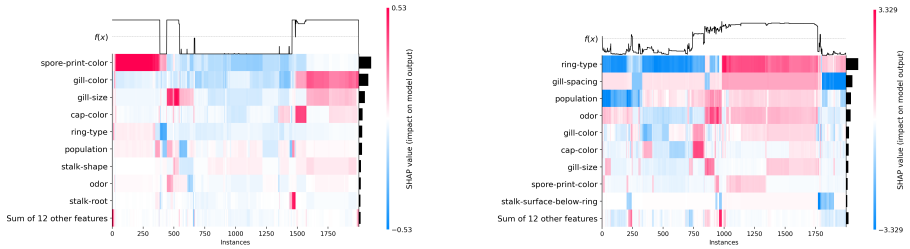
Fig. 15 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 15. SHAP beeswarm plot for special attributes have 5% noise and others have 0%

Fig. 16 shows the heatmap plots generated by SHAP from these two models.

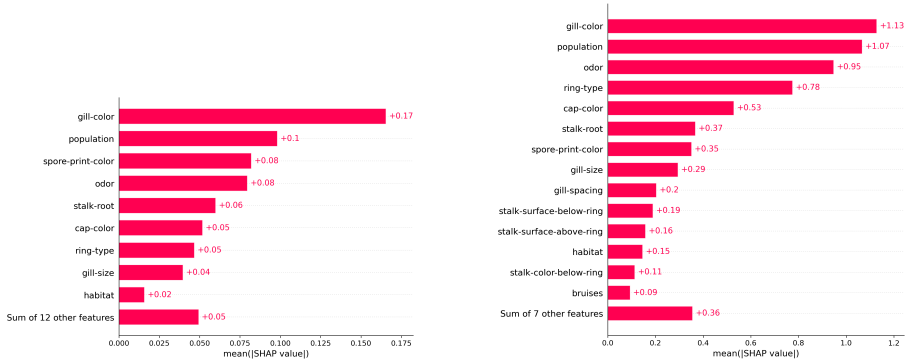


(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 16. SHAP heatmap plot for special attributes have 5% noise and others have 0%

From Fig. 14, Fig. 15 and Fig. 16, it can be seen that the rule-related attributes in the interpretation of the decision tree do not have a significant impact on the prediction, which suggests that the decision tree does not fit the custom rule well. In contrast, in the GBC interpretation, although it is affected by noise and the interpretation becomes complicated, it still reflects the fitting of the rules. Most rule-related attributes have a relatively large impact on the prediction. At this point, only the GBC's predictions can be trusted.

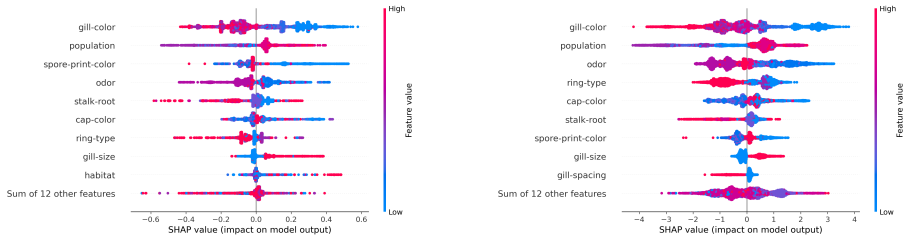
Special columns 10% others 5% Fig. 17 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 17. SHAP bar plot for special attributes have 10% noise and others have 5%

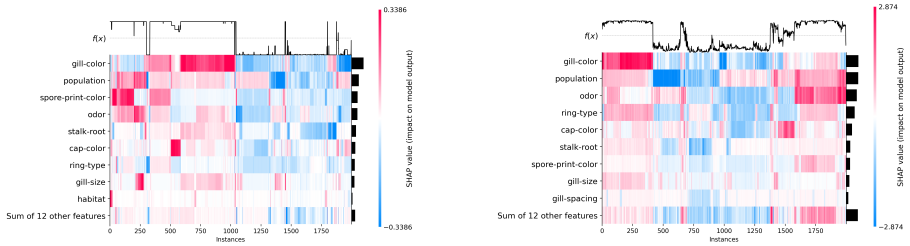
Fig. 18 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 18. SHAP beeswarm plot for special attributes have 10% noise and others have 5%

Fig. 19 shows the heatmap plots generated by SHAP from these two models.

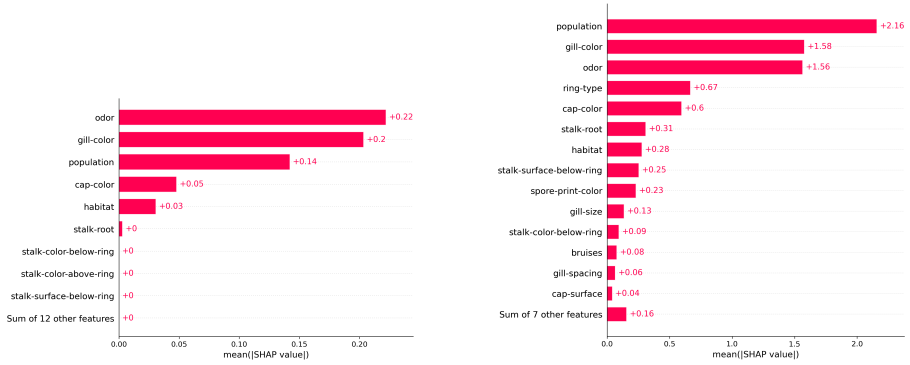


(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 19. SHAP heatmap plot for special attributes have 10% noise and others have 5%

From Fig. 17, Fig. 18,Fig. 19 , it is clear that both models roughly fit the custom rules, and the rule-related attributes are well utilised by the models. However, GBC has a higher F1-score.

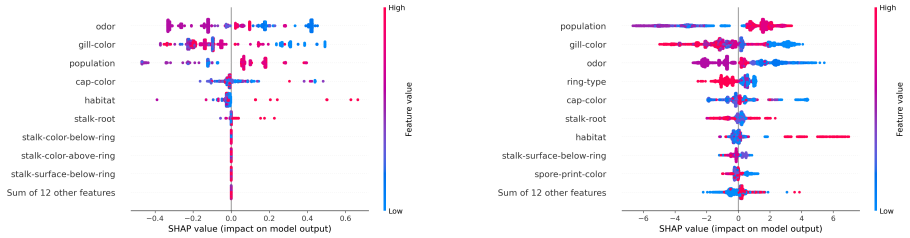
Special columns 0% others 5% Fig. 20 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 20. SHAP bar plot for special attributes have 0% noise and others have 5%

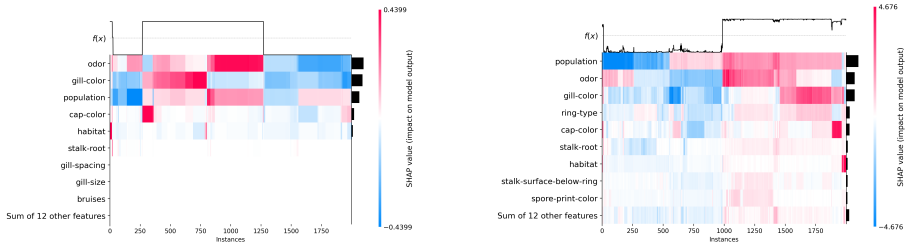
Fig. 21 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 21. SHAP beeswarm plot for special attributes have 0% noise and others have 5%

Fig. 22 shows the heatmap plots generated by SHAP from these two models.

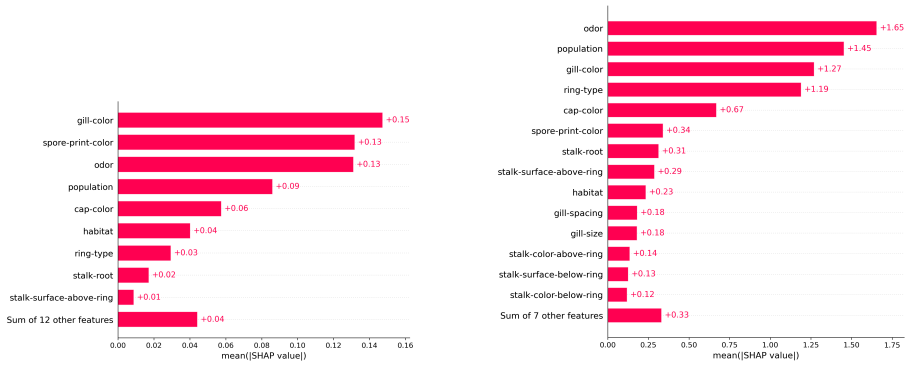


(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 22. SHAP heatmap plot for special attributes have 0% noise and others have 5%

As can be seen in Fig. 20,Fig. 21,Fig. 22, at this point, both models fit the custom rules well, due to the fact that the influence of the other attributes is reduced by the noise. At this point, the predictions of both models are trustworthy.

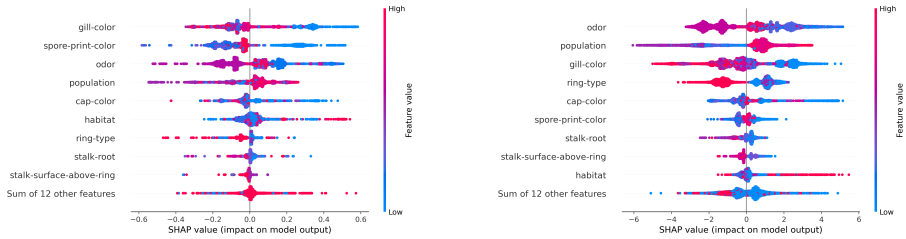
Special columns 5% others 10% Fig. 23 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 23. SHAP bar plot for special attributes have 5% noise and others have 10%

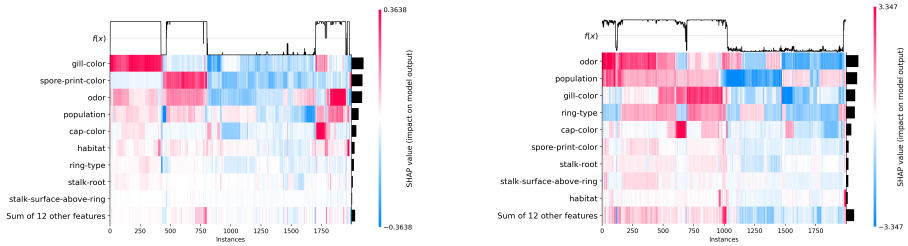
Fig. 24 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 24. SHAP beeswarm plot for special attributes have 10% noise and others have 5%

Fig. 25 shows the heatmap plots generated by SHAP from these two models.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 25. SHAP heatmap plot for special attributes have 5% noise and others have 10%

From Fig. 23, Fig. 24,Fig. 25 , it is clear that both models roughly fit the custom rules, and the rule-related attributes are well utilised by the models. However, GBC has a higher F1-score.So GBC is more trustworthy.

5.2 Stellar dataset

Same Perlin noise for all attributes The decision tree and GBC were trained separately on the stellar classification dataset, which contained different levels. As a result, the following outcomes were obtained:

Table 3 shows the performance of the two models on the stellar classification datasets with different noise level(same level on each attribute)

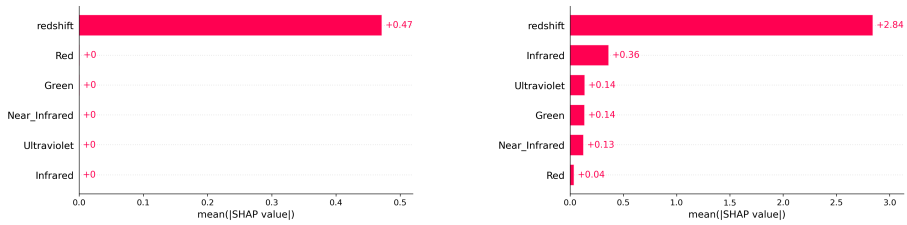
Table 3. F1-score from CV for same Perlin noise level on stellar dataset

Noise level	Result from DT	Result from GBC
No noise	1.000 +/- 0.000	1.000 +/- 0.000
Smooth	0.920 +/- 0.003	0.929 +/- 0.002
Medium	0.888 +/- 0.003	0.992 +/- 0.004
Rough	0.857 +/- 0.004	0.911 +/- 0.003

Where noise level 'Smooth' means less variation and smaller range of noise, 'rough' means more variation and bigger range of noise. As can be seen in Table 3, as the noise level rises, the F1-score of both models produces reductions. Among them, the decision tree received a greater impact compared to the GBC.

No noise Both models achieved a 100% F1-score on the noise-free dataset as shown in Table 3. The confirmation of whether the two models identified the defined rules can only be made through further analysis of SHAP's interpretation.

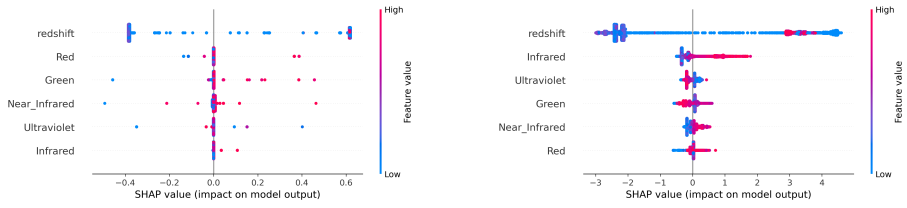
Fig. 26 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 26. SHAP bar plot for no noise on stellar dataset

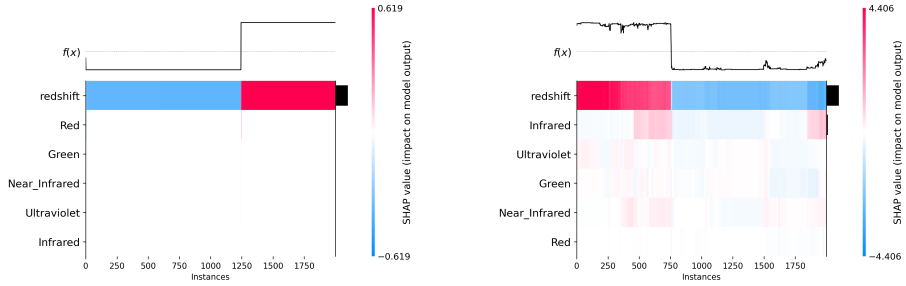
Fig. 27 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 27. SHAP beeswarm plot for no noise level on stellar dataset

Fig. 28 shows the beeswarm plots generated by SHAP.



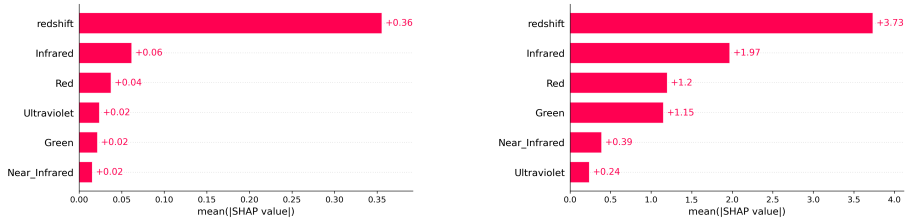
(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 28. SHAP heatmap plot for for no noise level on stellar dataset

As can be seen from Fig. 26, Fig. 27,Fig. 28, the decision tree's prediction strategy tends to use only the 'redshift' attribute, whereas the GBC's prediction strategy uses all the attributes, so the GBC's strategy fits the custom rule better. Also, since the average of the five filter data and 'redshift' work together to classify the classification in the custom rule, 'redshift' is the most important attribute. This is also reflected in GBC's prediction strategy.

Overall, although both models get very high F1-scores, GBC's training strategy fits the rules better, so GBC is more trustworthy.

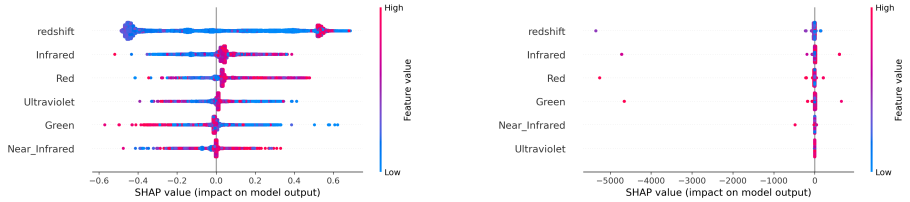
smooth perlin noise Fig. 29 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 29. SHAP bar plot for smooth Perlin noise on stellar dataset

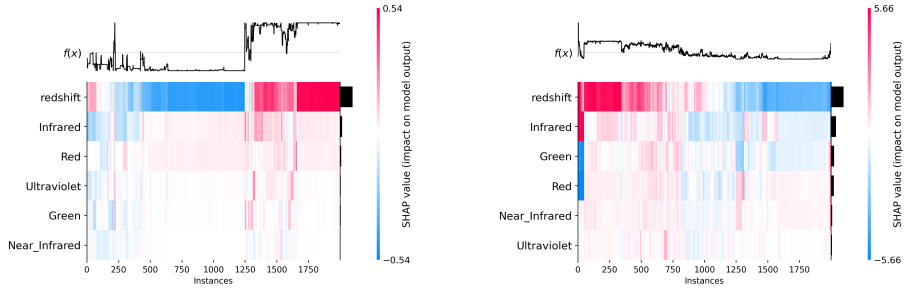
Fig. 30 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 30. SHAP beeswarm plot for smooth Perlin noise on stellar dataset

Fig. 31 shows the beeswarm plots generated by SHAP.



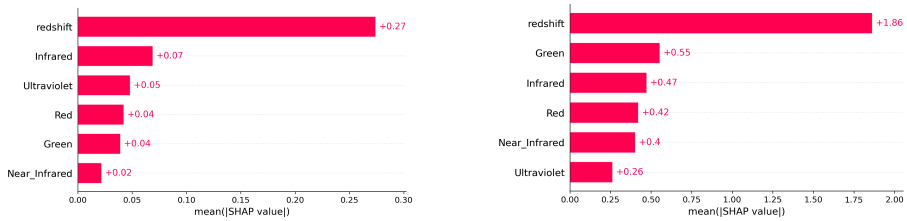
(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 31. SHAP heatmap plot for for smooth Perlin noise on stellar dataset

As can be seen from Fig. 29, Fig. 30,Fig. 31, both models use the full set of attributes, with 'redshift' as the most important attribute, which is a fit to a custom rule.

However, F1-score for GBC decreased less comparing with decision tree, so the GBC is more trustworthy.

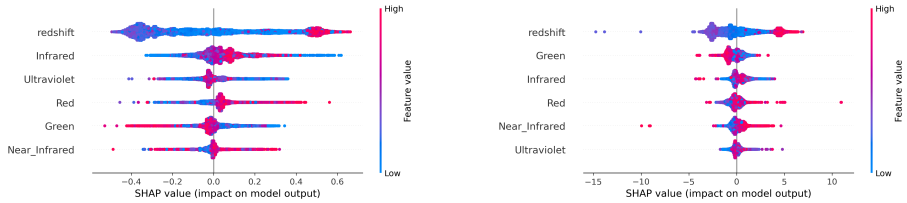
medium perlin noise Fig. 32 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 32. SHAP bar plot for medium Perlin noise on stellar dataset

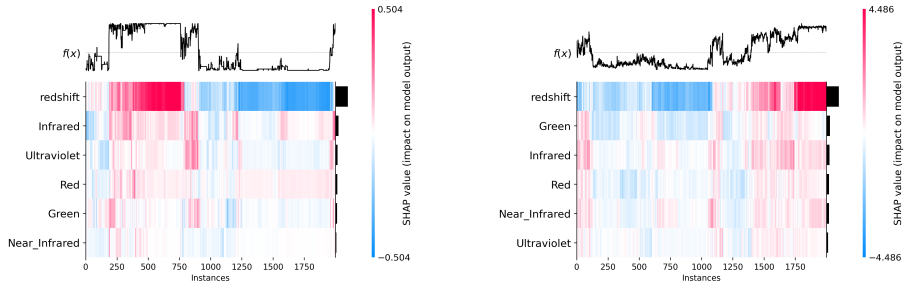
Fig. 33 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 33. SHAP beeswarm plot for medium Perlin noise on stellar dataset

Fig. 34 shows the beeswarm plots generated by SHAP.



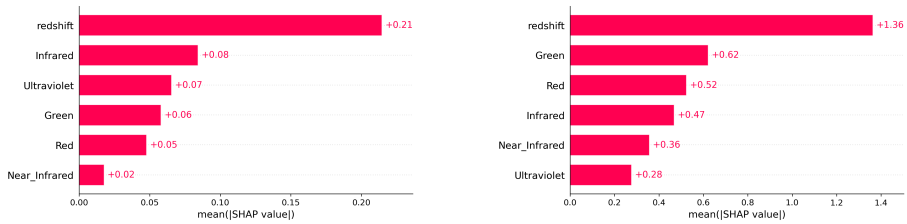
(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 34. SHAP heatmap plot for for medium Perlin noise on stellar dataset

As can be seen from Fig. 32, Fig. 33,Fig. 34, with the addition of 'medium' perlin noise, the dataset becomes complex and both models are trying to fit custom rules, but both are quite disturbed by the noise.

However, F1-score for GBC decreased less comparing with decision tree, so the GBC is more trustworthy.

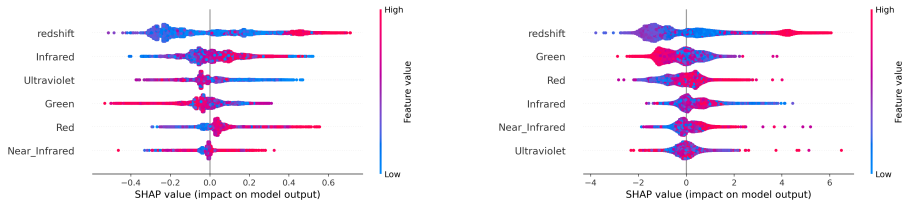
rough perlin noise Fig. 35 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 35. SHAP bar plot for rough Perlin noise on stellar dataset

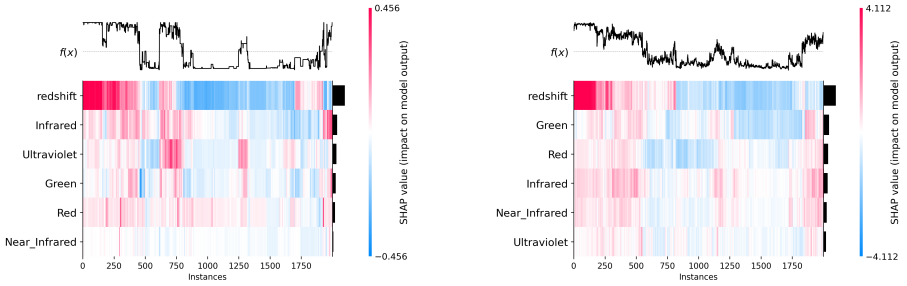
Fig. 36 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 36. SHAP beeswarm plot for rough Perlin noise on stellar dataset

Fig. 37 shows the beeswarm plots generated by SHAP.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 37. SHAP heatmap plot for rough Perlin noise on stellar dataset

As can be seen from Fig. 35, Fig. 36,Fig. 37, with the addition of 'rough' Perlin noise, the dataset becomes very complex and both models are trying to fit custom rules, but both are quite disturbed by the noise.

However, F1-score for GBC decreased less comparing with decision tree, so the GBC is more trustworthy.

Same Gaussian noise for all attributes The decision tree and GBC were trained separately on the stellar classification dataset, which contained different levels. As a result, the following outcomes were obtained:

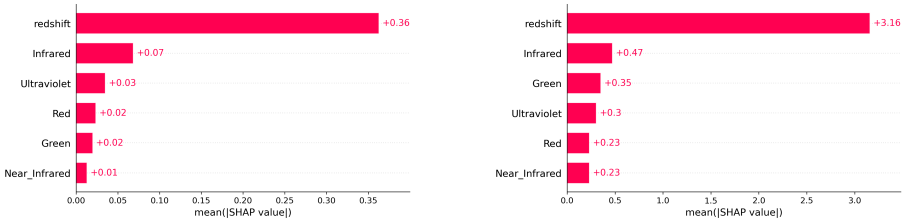
Table 4 shows the performance of the two models on the stellar classification datasets with different noise level(same level on each attribute)

Table 4. F1-score from CV for same Perlin noise level on stellar dataset

Noise level(std)	Result from DT	Result from GBC
0.1	0.923 +/- 0.003	0.953 +/- 0.001
0.2	0.888 +/- 0.002	0.928 +/- 0.001
0.3	0.843 +/- 0.004	0.894 +/- 0.002

Where noise level means the standard derivation of Gaussian distribution that generate the noise. As can be seen in Table 4, with noise level increasing, the F1-score of both two models decrease sharply. Among them, GBC got influenced by noise less.

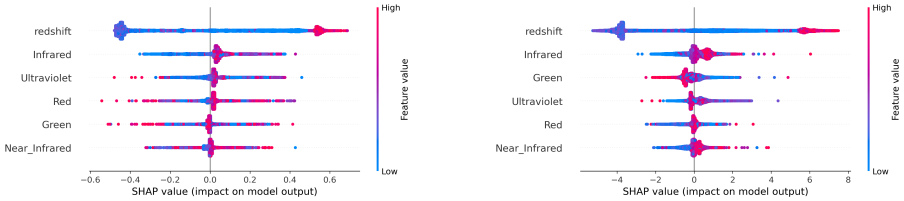
noise level 0.1 Fig. 38 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 38. SHAP bar plot for 0.1 Gaussian noise on stellar dataset

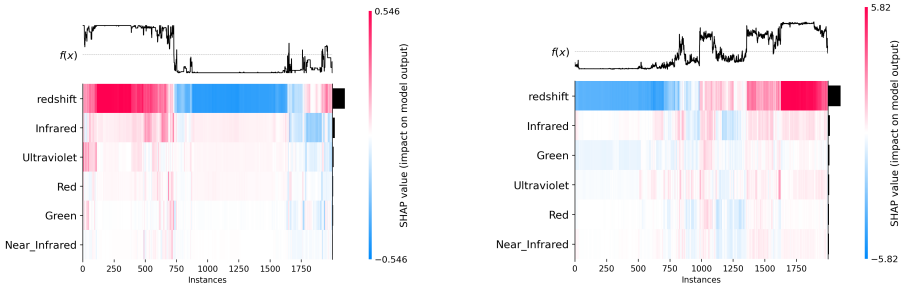
Fig. 39 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 39. SHAP beeswarm plot for 0.1 Gaussian noise on stellar dataset

Fig. 40 shows the beeswarm plots generated by SHAP.

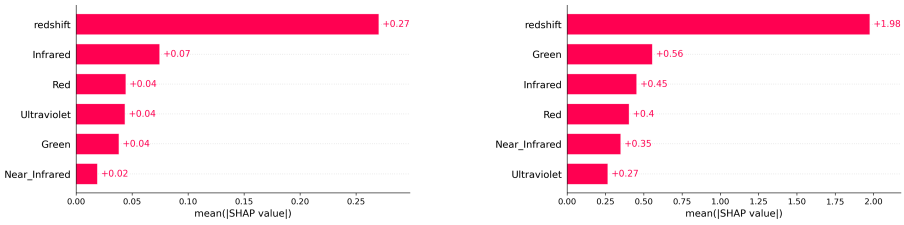


(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 40. SHAP heatmap plot for for 0.1 Gaussian noise on stellar dataset

As can be seen from Fig. 38, Fig. 39,Fig. 40, when Gaussian noise with a standard deviation of 0.1 was added to the dataset, both models were affected by the noise. The F1-score of GBC was less affected. So the GBC is more trustworthy.

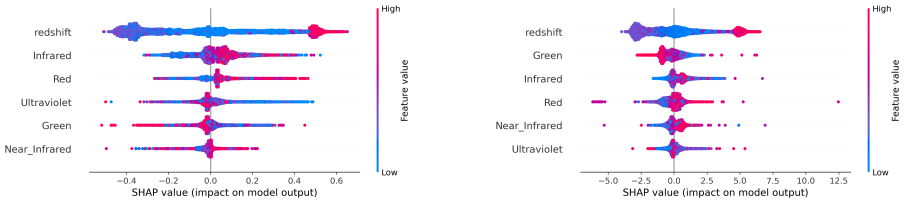
noise level 0.2 Fig. 41 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 41. SHAP bar plot for 0.2 Gaussian noise on stellar dataset

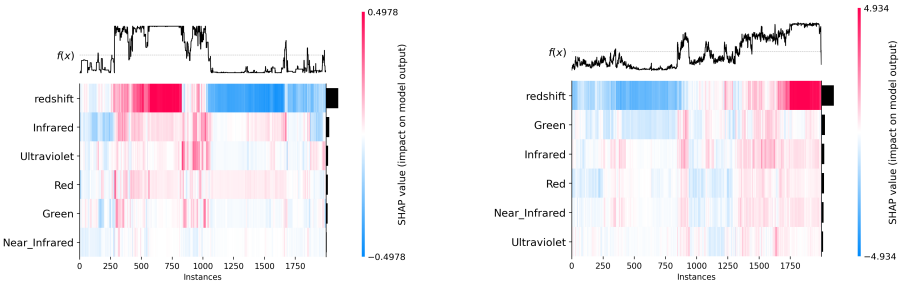
Fig. 42 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 42. SHAP beeswarm plot for 0.2 Gaussian noise on stellar dataset

Fig. 43 shows the beeswarm plots generated by SHAP.

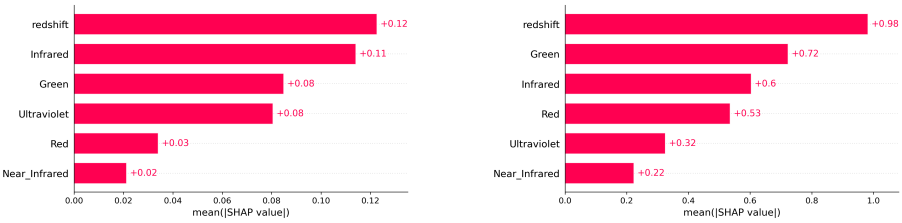


(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 43. SHAP heatmap plot for for 0.1 Gaussian noise on stellar dataset

As can be seen from Fig. 41, Fig. 42,Fig. 43, when Gaussian noise with a standard deviation of 0.2 was added to the dataset, both models were affected by the noise. The F1-score of GBC was less affected. So the GBC is more trustworthy.

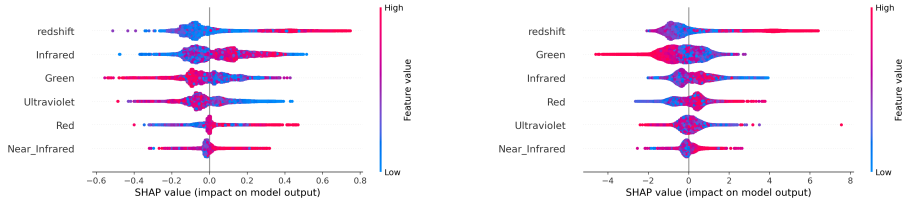
noise level 0.5 Fig. 44 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 44. SHAP bar plot for 0.5 Gaussian noise on stellar dataset

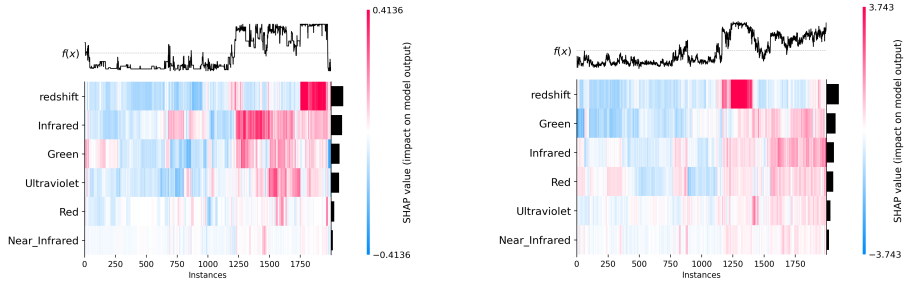
Fig. 45 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 45. SHAP beeswarm plot for 0.5 Gaussian noise on stellar dataset

Fig. 46 shows the beeswarm plots generated by SHAP.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 46. SHAP heatmap plot for for 0.5 Gaussian noise on stellar dataset

As can be seen from Fig. 44, Fig. 45,Fig. 46, when Gaussian noise with a standard deviation of 0.5 was added to the dataset, both models were severely affected by the noise. The F1-score of GBC was less affected. So the GBC is more trustworthy.

Different Perlin noise The decision tree and GBC were trained separately on the stellar classification dataset, which contained different levels. As a result, the following outcomes were obtained:

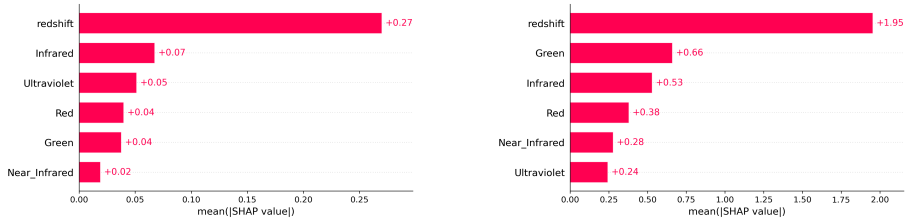
Table 5 shows the performance of the two models on the stellar classification datasets with different noise level.

Table 5. F1-score from CV for different noise level on stellar dataset

Normal columns	Special columns	Result from DT	Result from GBC
Smooth	Rough	0.884 +/- 0.004	0.927 +/- 0.001
Rough	Smooth	0.917 +/- 0.003	0.946 +/- 0.001

From Table 5, it can be seen that the F1-score of both models is perturbed by noise, but the model is less perturbed by noise when there is less noise in the 'redshift' (special attribute). Meanwhile, GBC is less affected by noise.

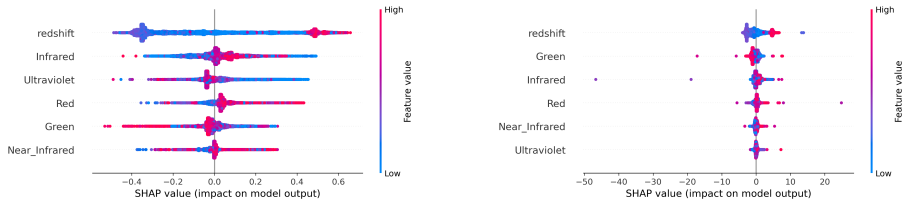
Special attributes have higher noise level Fig. 47 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 47. SHAP bar plot for special attributes have higher noise level on stellar dataset

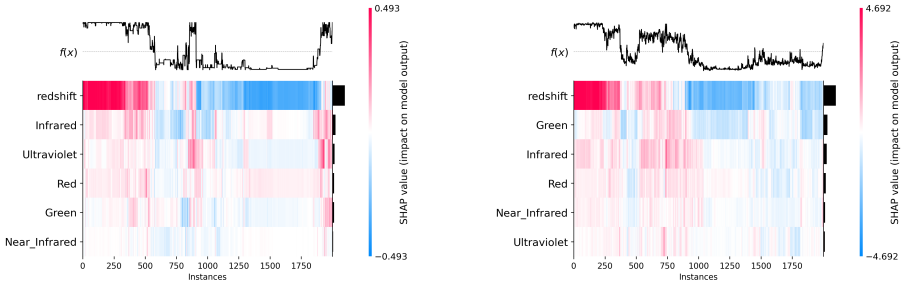
Fig. 48 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 48. SHAP beeswarm plot for special attributes have higher noise level on stellar dataset

Fig. 49 shows the beeswarm plots generated by SHAP.



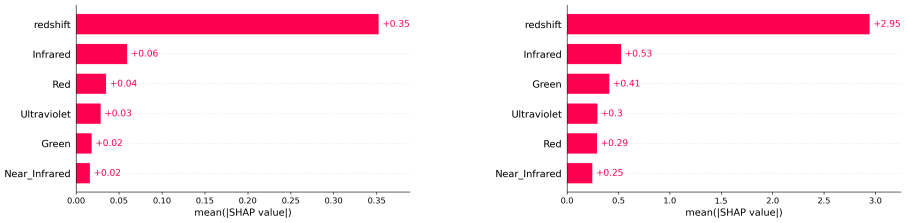
(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 49. SHAP heatmap plot for special attributes have higher noise level on stellar dataset

As can be seen from Fig. 47, Fig. 48,Fig. 49, with the addition of 'rough' Perlin noise, the dataset becomes very complex and both models are trying to fit custom rules, but both are quite disturbed by the noise.

However, F1-score for GBC decreased less comparing with decision tree, so the GBC is more trustworthy.

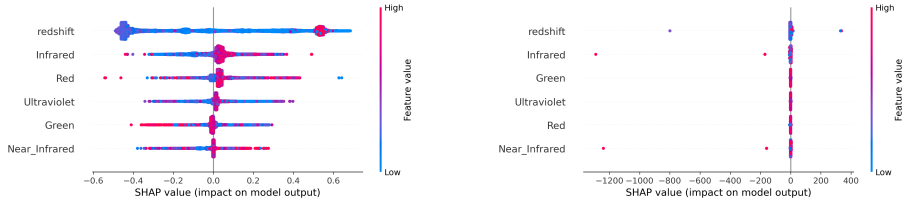
Special attributes have lower noise level Fig. 50 shows the bar plots generated by SHAP for decision tree and GBC. These bar plots show the ranking of feature importance of two models.



(a) Decision tree bar plot (b) GBC bar plot

Fig. 50. SHAP bar plot for special attributes have lower noise level on stellar dataset

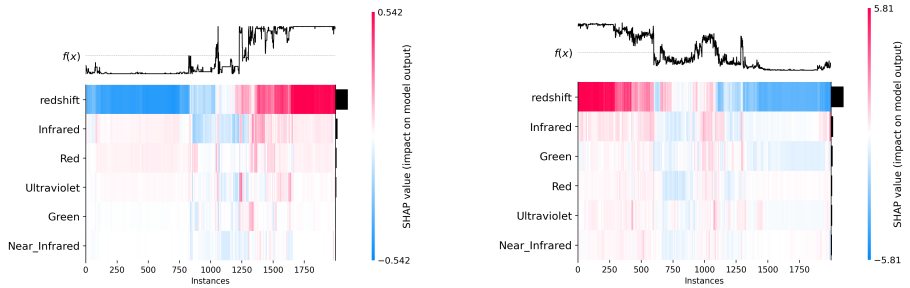
Fig. 51 shows the beeswarm plots generated by SHAP.



(a) Decision tree bee plot (b) GBC bee plot

Fig. 51. SHAP beeswarm plot for special attributes have lower noise level on stellar dataset

Fig. 52 shows the beeswarm plots generated by SHAP.



(a) Decision tree heatmap plot (b) GBC heatmap plot

Fig. 52. SHAP heatmap plot for special attributes have lower noise level on stellar dataset

As can be seen from Fig. 50, Fig. 51,Fig. 52, with the addition of 'rough' Perlin noise, the dataset becomes very complex and both models are trying to fit custom rules, but both are quite disturbed by the noise.

However, F1-score for GBC decreased less comparing with decision tree, so the GBC is more trustworthy.

6 Summary

The performance of the two models was compared very intuitively by analysing multiple experiments on both datasets. The performance is divided into three areas, the F1-score derived from model training, the interpretation of the model, and the ability of the model to cope with noise.

When there is no noise, both models can achieve very high F1-scores on both datasets.However, GBC's prediction strategy tends to fit custom rules bet-

ter, whereas the decision tree prefers to classify using only a small number of attributes.

When noise is added to the dataset, the prediction results and prediction strategies of both models are perturbed by noise. In contrast, GBC is less affected by noise, while decision trees are more sensitive to noise. In addition, when the noise is small and the dataset is simple, the model can find the custom rules hidden in the dataset. Whereas, when the noise is too large, the dataset becomes complex and challenging and the model often fails to find the custom rules correctly.

Overall, in a very large number of experiments on two datasets have very different styles, the more complexly structured GBC tends to outperform the decision tree. In other word, the prediction made by GBC is more trustworthy. When assessing model performance, not only the cross-validation scores and performance metrics, but also the model's explanations should be considered.

7 Future work

In the experiments in this paper, there are still many deficiencies that can be optimised.

Firstly, due to the limited computational power of the experimental environment, there is a tendency in the experiments to sample representative datasets when generating heat maps. The problem with this is that the quality of the sampling affects the quality of the interpretation to some extent. If sufficient computational power is available, heat maps of the complete dataset should be generated for a better interpretation of the model.

Secondly, when processing the mushroom dataset, the data is encoded as numbers. In the experiments of this paper, the default encoding of the encoder is used. The special attribute values in the rules can be distinguished from other attribute values when encoding, then the interpretation of the beeswarm map generated by SHAP can be better analysed.

Finally, due to TreeSHAP's lack of support for GBC on multiclassification problems, the rules in the experiments are all binary classification problems. The performance and interpretation of the model on multiclassification problems still need further research.

References

1. Mushroom. UCI Machine Learning Repository (1987), DOI: https://doi.org/10.24432/C5959T
2. Adadi, A., Berrada, M.: Peeking inside the black-box: A survey on explainable artificial intelligence (xai). IEEE Access **6**, 52138–52160 (2018). https://doi.org/10.1109/ACCESS.2018.2870052
3. et al., A.: The seventeenth data release of the sloan digital sky surveys: Complete release of manga, mastar, and apogee-2 data. The Astrophysical Journal Supplement Series **259**(2), 35 (mar 2022). https://doi.org/10.3847/1538-4365/ac4414, https://dx.doi.org/10.3847/1538-4365/ac4414

4. Alvarez-Melis, D., Jaakkola, T.S.: On the robustness of interpretability methods (2018)
5. Arrieta, A.B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al.: Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Information fusion **58**, 82–115 (2020)
6. Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning (2017)
7. Fedesoriano: Stellar classification dataset - sdss17 (Jan 2022), https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17
8. Gauss, C.F.: Theoria motus corporum coelestium in sectionibus conicis solem ambientium, vol. 7. FA Perthes (1877)
9. Green, S.: Implementing improved perlin noise. GPU Gems **2**, 409–416 (2005)
10. Gupta, S., Gupta, A.: Dealing with noise problem in machine learning data-sets: A systematic review. Procedia Computer Science **161**, 466–474 (2019). https://doi.org/https://doi.org/10.1016/j.procs.2019.11.146, https://www.sciencedirect.com/science/article/pii/S1877050919318575, the Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia
11. Jain, A.K.: Fundamentals of digital image processing. Prentice-Hall, Inc. (1989)
12. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton, A., Robert, C.C. (eds.) Proceedings of the 19th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 51, pp. 240–248. PMLR, Cadiz, Spain (09–11 May 2016), https://proceedings.mlr.press/v51/jamieson16.html
13. Lundberg, S.M., Erion, G.G., Lee, S.I.: Consistent individualized feature attribution for tree ensembles (2019)
14. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 4765–4774. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf
15. Molnar, C., Casalicchio, G., Bischl, B.: Interpretable machine learning – a brief history, state-of-the-art and challenges. In: ECML PKDD 2020 Workshops, pp. 417–431. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-65965-3_28, https://doi.org/10.1007/978-3-030-65965-3_28
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
17. Perlin, K.: An image synthesizer. ACM Siggraph Computer Graphics **19**(3), 287–296 (1985)
18. Ribeiro, M.T., Singh, S., Guestrin, C.: " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1135–1144 (2016)
19. Ryden, B., Peterson, B.M.: Foundations of astrophysics. Cambridge University Press (2020)
20. Saseendran, A.T., Setia, L., Chhabria, V., Chakraborty, D., Barman Roy, A.: Impact of noise in dataset on machine learning algorithms. In Machine Learning Research pp. 0–8 (2019)

21. Shapley, L.S.: 17. a value for n-person games. In: Contributions to the Theory of Games (AM-28), Volume II, pp. 307–318. Princeton University Press (Dec 1953). https://doi.org/10.1515/9781400881970-018, https://doi.org/10.1515/9781400881970-018
22. Stone, M.: Cross-validatory choice and assessment of statistical predictions. Journal of the Royal Statistical Society: Series B (Methodological) **36**(2), 111–133 (1974). https://doi.org/https://doi.org/10.1111/j.2517-6161.1974.tb00994.x
23. Zhu, X., Wu, X.: Class noise vs. attribute noise: A quantitative study. Artificial Intelligence Review **22**(3), 177–210 (Nov 2004). https://doi.org/10.1007/s10462-004-0751-8, https://doi.org/10.1007/s10462-004-0751-8