Using SHAP Post-Model Explainability as a Model-Agnostic Feature Selection Technique

Joshua Gottlieb
Seidenberg School of CSIS
Pace University
New York City, USA
jg05394n@pace.edu

Martin Sichali Chunsen
Seidenberg School of CSIS
Pace University
New York City, USA
cs83339n@pace.edu

Alexander Yoo
Seidenberg School of CSIS
Pace University
New York City, USA
ay71173n@pace.edu

Advisor: Dr. Krishna Bathula
Seidenberg School of CSIS
Pace University
New York City, USA
kbathula@pace.edu

***Abstract*—Feature selection techniques in data science help reduce dimensionality of datasets, remove sparsity, and eliminate noisy patterns, thereby improving generalization of models to future data. Many feature selection techniques fail to optimize for the predictive power of the model, require iterating through the extensive feature space, or apply only to certain model types. SHAP values focus on capturing the predictive power of each feature within the context of the trained model in order to provide explainability. Because SHAP treats the model as a black box, the SHAP technique is model agnostic. In this paper, we employ SHAP values to capture and rank the important features learned by a model and to use these rankings for feature selection. We build upon prior work using SHAP as a feature selection technique by creating the MAX and SUM selection algorithms, which are designed to align with intuitive feature selection goals without the requirement for costly iterative processes. We validate the effectiveness of our SUM and MAX SHAP selection algorithms by testing their performance with five machine learning models using ten diverse datasets, demonstrating that these SHAP selection strategies produce equivalent or superior model performance and greater feature reduction compared to other state-of-the-art techniques.**

***Index Terms*—Feature selection, model-agnostic methods, Shapley values, machine learning.**

I. Introduction

High dimensionality in datasets is a common occurrence and presents many issues. Not all features are relevant to the machine learning task, and higher dimensionality leads to an increase in noisy and erroneous patterns within the dataset. Data also becomes sparse in higher dimensions, requiring increasingly more data to achieve acceptable performance as dimensionality grows [1]. These issues lead to a decrease in model performance, as models fit themselves to irrelevant patterns and generalize poorly to new data, underperforming due to high dataset complexity.

There are projective techniques, such as PCA [2], t-SNE [3], or UMAP [4], which take high dimensional data and directly project it into a lower dimensional space. However, these dimensionality techniques have the unfortunate consequence of distorting or destroying all model interpretability due to combining features together in ways which obscure the effects of each contributing feature. This outcome is not ideal for most modeling scenarios, where it is necessary not only that a model works but to be able to understand why the model works and to pinpoint which features are most useful for the real-world task being modeled.

Feature selection is the alternative to dimensionality reduction techniques and works by selecting a subset of the features instead of using the entire feature set. By using each feature in its original form, model interpretability is preserved while reducing data dimensionality. In order to select features, a selection criteria (i.e., a metric) [5] is needed to rank features and find the optimal subset. In general, it is not feasible to test every possible feature subset, as the number of feature subsets for a dataset with n features is 2^n.

Existing feature selection techniques are divided into three groups: filter, wrapper, and embedded [5], [6]. Each of these groups has drawbacks when it comes to how they select feature subsets. Filter methods are unable to utilize the patterns learned by a model to select features; wrapper methods are computationally expensive and require a complex search strategy to test feature subsets; and embedded methods are model-specific and cannot be used across models. Instead of traditional feature selection techniques, it is possible to use a technique from the field of model explainability known as SHAP [7]. SHAP is a model-agnostic technique which attempts to explain black box models by assigning a contribution value for each feature at each data point. These SHAP values can be used as a proxy for feature importance and are the basis of the work produced in [8].

An important step in feature selection is the ultimate choice of k features to use. Prior work with SHAP either failed to address the choice of k or selected k through computationally expensive iterative algorithms. Our paper expands upon prior work using SHAP values for feature selection by creating two new algorithms, MAX and SUM, for selecting features which require no iteration. The MAX strategy filters features by keeping features that are only as strong as a user-specified proportion of the maximum feature importance. The SUM strategy selects features in order of importance until a user-specified proportion of the summed feature importances is captured. We designed these strategies to align with human intuition for feature selection. The MAX strategy frames feature selection in terms of relative feature strength, while the SUM strategy frames feature selection in terms of capturing a meaningful amount of learned feature importance. Both of

these strategies are explained in greater detail in section IV.

We tested our work using five different models and repeated our experiments across ten publicly available datasets to ensure that our results are reproducible under different conditions and are not biased by particularly favorable conditions inherent to any single dataset.

The remainder of this paper is organized as follows. Section II presents an overview of feature selection methods and of SHAP explanation algorithms. Section III briefly covers previous work using SHAP values for feature selection and identifies the gaps and drawbacks present in prior work that our work aims to address. Section IV outlines the methodology of our experiments. Section V contains the analysis and results of our experiments, while Section VI provides a succinct conclusion of our findings and suggestions for further work. The Appendix covers technical details regarding hyperparameter choices for models and feature selection techniques.

II. Technical Background

A. Feature Selection Methods

The goal of feature selection methods is to find the ideal subset of features that are the most useful and relevant to the chosen machine learning task. Features may be relevant, irrelevant, or redundant [6], and different feature selection techniques tackle the problem of choosing only the relevant features in different manners. For a dataset with n features, the total possible number of feature subsets is 2^n, making it infeasible to test all possible feature subsets to find the optimal feature subset for most datasets. The ideal feature selection technique chooses only the best and most relevant subset of features; however, in practice, there is a trade-off between the efficiency and simplicity of the feature selection technique and its effectiveness in finding the best feature subset [1]. There are three main categories of feature selection methods: filter, wrapper, and embedded methods [5], [6].

Filter methods employ a performance measure in order to rank features by importance and relevance. Filter methods are not reliant on the final algorithm used for modeling and as such are model-agnostic. Filter methods can be further broken into categories based on their scoring criteria, such as distance-based, information-based, and statistically-based. Filter methods may be univariate or multivariate, determining whether they are capable of picking up feature interactions. Some examples of filter methods are Pearson Correlation and Fast Correlation-Based Filter [9], which filter based on target correlations; Mutual Information [9], Minimum Redundancy Maximum Relevance (mRMR) [10], and Symmetrical Uncertainty [11] which measure the information of features with the target and with each other; and ReliefF [12] and Fisher Score [13] selection which use distance measures between instances to rank features. Filter methods are typically scalable and efficient, as they rely on simple calculations to score the features [6]. However, they are unable to leverage the patterns found during modeling and may result in less performant feature subsets than other methods [14].

Wrapper methods consider the effectiveness of feature subsets by employing a modeling algorithm as a black box evaluator on each subset until finding an optimal subset [6]. Each subset is tested using the chosen model and scored using an appropriate metric, such as, F1 score, and the subset with the best performance is used to train the final model. Wrapper methods require a search strategy in order to effectively explore the feature subsets, which can range from as simple as greedy forward selection or backward elimination to as complex as genetic algorithms and particle swarm optimization [6], [15], [16]. While wrapper methods do tend to select subsets with higher performance than filter based methods [14], they have much higher computational complexity due to the need to iteratively fit models to the data [9]. It is typically only feasible to use simple algorithms as the black box model, such as Naive Bayes or Linear SVM, and there is no guarantee that the subsets found by these models will generalize to the final model used [6].

Embedded methods perform feature selection during or after model training. These methods are either built in to the model itself or are added extensions to the model algorithms. Some common algorithms include CART and C4.5 for decision trees, which control the complexity of the trees by pruning isolated splits or imposing constraints on maximum tree depth [6]. Other common implementations, such as Lasso or ElasticNet, impose regularization penalties to prevent large coefficients and to force small coefficients towards zero, performing feature selection by removing the effect of irrelevant features. A benefit of embedded methods is that they are more efficient than wrapper methods since they do not require iterative retraining of the model, and unlike filter methods, they are able to utilize patterns found by the model itself. However, most embedded methods are model-specific and thus cannot be used across all model types.

B. SHAP Explainability

Shapley Additive Explanations (SHAP) is considered a state-of-the art method for model-agnostic interpretation of black box models. SHAP values are an additive model explanation approach, based on Shapley values from cooperative game theory [7], [17]. SHAP values are typically calculated locally at each data point. Each feature is assigned an importance value that measures how much that feature contributes to the conditional expectation that influences the local data point away from the global expected prediction. For classification tasks, the SHAP values represent the influence of each feature in pushing the classification of a data point towards each class compared to the average class prediction of all data points. The sum of all of the SHAP values equals the original classification by the model. SHAP values satisfy many desirable properties, including local accuracy, missingness, and consistency [7].

While SHAP values are often used to explain individual predictions, they can be used to approximate the global black box model [18]. By taking the absolute value of the SHAP values for each data point and averaging them, the global feature importance for the model is attained [17]. Each of these

mean absolute SHAP values provides a global importance of how much the feature influences predictions away from the mean prediction for the entire dataset. This usage of SHAP is used for model explainability, but conveniently this process produces a feature ranking. By sorting these mean absolute SHAP values in decreasing magnitude, the features are scored in a method similar to filter methods while using patterns learned from the model itself, akin to an embedded method. We use this fact as the basis for our feature selection technique. Since SHAP provides a method of interpreting the important features for predictions, it is reasonable to assume that the most important features, as ranked by SHAP, would make for the strongest feature subset.

One important problem to note is that the calculation of SHAP values is known to be an NP-hard problem [19], as sampling all of the possible conditional probabilities for the Shapley coalitions requires iterating over the same feature subspace of size 2^n discussed earlier. There are several SHAP implementations that seek to solve this problem, including KernelSHAP, PermSHAP, and TreeSHAP. It is beyond the scope of this paper to cover each of these algorithms, but the main consideration is the trade-off between speed, model-specificity, and ability to capture feature interactions. KernelSHAP is a model-agnostic approach that fits a linear regression to the SHAP values, and as such it cannot capture any feature interactions [7], [17]. PermSHAP is also a model-agnostic approach that samples the features in forward and reverse directions using antithetic sampling [19]. This process can be repeated any number of times, but by doing this process exactly once, evaluating the model function $2n + 1$ times per background data sample, up to second-order interactions can be captured [20]. PermSHAP is generally more computationally efficient than KernelSHAP in practice [17]. TreeSHAP utilizes the structure of tree-based algorithms to greatly reduce the complexity of calculating SHAP values. TreeSHAP calculates the exact SHAP values with all feature interactions in $O(TLD^2)$ time, where T is the number of trees, L is the maximum number of leaves in any tree, and D is the maximum depth of any tree [17], [21]. While this is an incredible performance increase over the other discussed implementations, TreeSHAP is specific only to tree-based models.

Due to its model-agnostic nature and ability to capture feature interactions, we have elected to use the PermSHAP algorithm for our experiments. See the details on the Permutation explainer parameters as given in the Appendix.

III. Literature Review

The usage of SHAP values for feature selection is a relatively new concept. In [8], the TreeSHAP algorithm was utilized along with XGBoost models across eight small datasets and compared with ANOVA, Mutual Information, and Recursive Feature Elimination feature selection strategies. The strategies employed in [8] are similar to the strategies we propose in this paper; however, by using TreeSHAP, the versatility of SHAP selection for non-tree based models was not tested. In addition, [8] offered no particular strategy for choosing the number of features to keep when using SHAP.

In [22], TreeSHAP was utilized for regression tasks. However, the SHAP selection was combined with a sequential forward selection strategy, requiring many iterations to find the optimal feature subsets, which removes the benefit of using SHAP as a one-pass technique for feature selection.

TreeSHAP and XGBoost models were used in [23] to predict credit card fraud. Rather than use the features as sorted by SHAP values, an additional step of assigning statistical significance to each feature importance was performed. For classification tasks, logistic regression was used on the SHAP values, while for regression tasks, linear regression was used. These feature significance tests were recursively repeated until a final feature set was selected. This paper claims that their algorithm produces better results with fewer features than the naive implementation of SHAP values for feature selection, but the authors only tested a single k value for the number of selected features for the naive implementation, and their algorithm produces only a slight improvement over the naive implementation while requiring three times the runtime. As this paper uses TreeSHAP, the results do not generalize to non-tree based models.

The naive implementation of SHAP values for feature selection was implemented in [24] for predicting credit card fraud. Although several model types were used, such as XGBoost, Decision Trees, CatBoost, Extremely Randomized Trees, and Random Forests, all models were tree-based and utilized TreeSHAP. In addition, the authors utilized a naive selection process, choosing the top 3, 5, 7, 10, or 15 features. Many of their results were inconclusive.

The paper in [25] compared the effectiveness of SHAP selection versus Lasso selection on a breast cancer dataset. Unlike most implementations of SHAP selection, the authors did not use mean absolute SHAP values and instead used raw SHAP values. They selected the top 20 most common features among SHAP explanations for seven different model types and used this feature set for selection, disconnecting the SHAP selection from a specific model type. This paper utilized many model types, including non-tree based models, such as Logistc Regression; however, we were unable to determine which particular SHAP algorithm was used within the paper.

The GitHub repository in [26] proposes the BorutaSHAP algorithm. This repository came to our attention due to being mentioned in passing within [27]; however, it has no associated paper itself. This algorithm is based upon the Boruta features selection algorithm [28] where the feature set is extended with "shadow" copies of each feature. These shadow copies are permuted to remove all correlations with the target, and a model is fit on the extended feature set. Features which perform worse than the shadow features are eliminated along with their shadow counterparts, and this process is performed iteratively until all shadow features have been eliminated. The BorutaSHAP algorithm expands upon the Boruta algorithm by using SHAP values for scoring. The iterative nature of this algorithm is both a blessing and a curse;

while it may lead to more confidence regarding the chosen feature subset, the iterative nature leads to a less efficient and more computationally intensive algorithm. In addition, the particular implementation of this model uses TreeSHAP as the underlying SHAP explainer, limiting it to only tree-based models.

The Powershap algorithm proposed in [27] is similar to the BorutaSHAP algorithm already described. However, instead of creating an entire shadow feature set, a single feature is randomly selected to use as a shadow copy during each iteration. This random feature changes for each iteration and after all iterations have been processed, each feature is ranked based on the percentage of iterations where it performed better than the random feature using one-tailed student-t tests. Any feature which is considered significant by this t-test is kept and all others are discarded. The Powershap algorithm is flexible and can be used with more models than just tree-based models; however, it suffers from similar issues as the BorutaSHAP algorithm due to it being an iterative process and being overall more computationally intensive than a one-pass SHAP selection technique.

We used the same one-pass naive algorithm as implemented in [8]. However, unlike many prior experiments, we have used two novel selection strategy implementations in order to try to find optimal feature subsets using a one-pass technique. We also have tested the robustness of our selection strategies by using the model-agnostic PermSHAP explainer and by repeating our experiment across datasets of varying complexities to evaluate the effectiveness and consistency of SHAP selection across different model types without relying on specific model architectures or underlying data patterns. The details of the MAX and SUM selection strategies, as well as the datasets used are presented in the following section.

IV. Methodology

A. The MAX and SUM Selection Strategies

The typical method of evaluating SHAP global explanations is to calculate the mean absolute SHAP explanations for each feature. Given a collection of local SHAP explanations ϕ_j for feature f_j, the global SHAP explanation I_j for feature f_j is calculated as the mean absolute sum of ϕ_j across all n samples:

$$I_j = \frac{1}{n}\sum_{i=1}^{n} |\phi_j(i)| \tag{1}$$

By ordering these global SHAP feature explanations I_j in descending order, a feature ranking is created among the m features. Ranking the features is only the first step in the feature selection process, as it is necessary to decide how many features to use for selection, commonly denoted as the variable k. Some papers using SHAP naively selected an arbitrary number of features to keep [8], [23], [24], [25]. Other papers relied on iterative algorithms of varying complexity to find the optimal value for k, resulting in high computational costs, especially if generalized to model-agnostic SHAP explainers [22], [26], [27]. We believe that the former method is unsatisfactory, given that the arbitrary choices for k were effectively random. The latter methods may be more robust, but they come at the cost of efficiency, and we believe that they lack a certain amount of elegance and intuitive simplicity. Given that SHAP values come from the field of machine learning explainability, it is important that the selection process is equally explainable.

To this end, we have created two selection strategies that allow the user to select features using meaningful metrics while working solely with the SHAP explanations generated from a one-pass process. These selection strategies were named the MAX and SUM strategies, based on the guiding metric used.

The MAX strategy involves selecting features based on the maximum feature strength discovered during the modeling process. Let the strongest feature strength be assigned as M. Note that the strongest feature strength is the first feature strength I_0 when the feature strengths are sorted in the traditional descending order.

$$M = \max_j(I_j) = I_0 \tag{2}$$

The user defines a proportion constant ρ in $[0, 1]$. Then, the features f_j are selected for the reduced feature set F if they have feature strength I_j at least as strong as $\rho \cdot M$:

$$F = \{f_j \mid j \in [0, m] \ \wedge \ I_j \geq \rho \cdot M\} \tag{3}$$

Because SHAP values represent the influence of a feature away from the average model prediction, the MAX strategy optimizes feature selection by choosing only features with relatively strong influences. This algorithm differs from simply selecting the top $x\%$ of features, because in datasets with highly imbalanced feature importances, it is possible for the strongest feature to dwarf the other feature importances. By ensuring each feature is at least $\rho \cdot M$ in strength, the user has defined what a meaningful threshold is for feature irrelevance in an intuitive and explainable manner. The strictness of the MAX strategy increases as ρ increases, since more features are added at lower proportions of the max feature strength. The tested values for ρ in our experiments were 0.01, 0.05, 0.1, 0.15, 0.25, and 0.5. The pseudocode for the MAX strategy is outlined in Algorithm 1.

Algorithm 1 MAX SHAP Psuedocode

```
Sort the feature strengths I_j in descending order: I_j ≥ I_{j+1}
Choose M = max(I_j) = I_0
Define a proportion constant ρ in [0, 1].
Initialize an index j = 0.
while I_j ≥ ρ · M do
    Increment j
end while
return [f_0, ..., f_j], the selected sorted features.
```

In contrast, the SUM strategy selects features based on the sum of feature importances discovered during the modeling process. Let S be the sum of feature importances.

$$S = \sum_j I_j \tag{4}$$

The user defines a proportion constant r in $[0, 1]$. A minimum number of features is selected for the reduced feature set F such that the sum of the feature importances is at least as great as $r \cdot S$:

$$F = \left\{ f_j \mid j \in [0, k], k = \min_{q \in [0,m]} \sum_{i=0}^{q} I_i \geq r \cdot S \right\} \quad (5)$$

In other words, the SUM strategy optimizes feature selection by choosing features with a combined contribution that is a sufficiently strong portion of the total contributions found. The SUM strategy is similar to a method commonly used in PCA dimensionality reduction, where components are added until a desired combined explained variance ratio is achieved. This algorithm also differs from selecting the top $x\%$ of features, as large portions of the discovered feature importances may be contained within few features. The SUM strategy ensures that enough features are kept to capture at least $r \cdot S$ of the total discovered feature importances, allowing the user to define a meaningful threshold for total feature explainability. The strictness for the SUM strategy decreases as r increases, since more features are added as the desired sum proportion increases. The values tested for r in our experiments were 0.5, 0.6, 0.7, 0.8, 0.9, and 0.95. The pseudocode for the SUM strategy is outlined in Algorithm 2.

Algorithm 2 SUM SHAP Pseudocode

Sort the feature strengths I_j in descending order: $I_j \geq I_{j+1}$
Calculate the total sum $S = \sum I_j$
Define a proportion constant r in [0, 1].
Initialize a running sum s and an index $j = 0$.
while $s < r \cdot S$ **do**
  Add I_j to s and increment j
end while
return $[f_0, \ldots, f_j]$, the selected sorted features.

Since the SUM and MAX strategies work with the raw SHAP feature strengths I_j, they require no iterative processes. In fact, by ordering the feature strengths, the algorithms can be optimized using vectorized NumPy operations and boolean masks. Thus, these algorithms incur minimal overhead beyond the initial calculation of SHAP values, a necessary cost incurred by any technique utilizing SHAP for feature selection.

B. Overview of Experimental Processes

The general SHAP selection process is outlined in the project architecture in Fig. 1. First, the dataset is preprocessed in preparation for training. Then, the model is trained and hyperparameter tuned on the full processed training dataset. These "full" models are used to calculate SHAP values with PermSHAP, which are then aggregated and sorted in descending order to create feature rankings. A feature subset is selected using either the SUM or MAX selection strategy. A new model is retrained on the reduced dataset using the same hyperparameters found during the tuning process. Finally, the performance of the models on the full and reduced datasets is compared.

TABLE I
DATASET INFORMATION

Dataset Name	Source	Instances	Features
Indian Liver Patient [29]	UCI	584	10
Heart Disease [30]	UCI	303	13
Mushroom [31]	UCI	8,124	22
SPECT Heart [32]	UCI	267	23
Breast Cancer [33]	UCI	569	30
Secondary Mushroom [34]	UCI	61,068	20
Credit Card Fraud [35]	Kaggle	284,807	30
Phishing URL [36]	UCI	235,795	54
Patient Survival [37]	Kaggle	91,700	84
Android Permissions [38]	UCI	29,333	86

C. Datasets Used and Preprocessing

Ten datasets were selected for use in this project. Eight of these datasets come from the UCI Machine Learning Repository, and the remaining two datasets are from Kaggle. These datasets were chosen due to their use in prior papers about feature selection and to test the effectiveness of SHAP across a variety of dataset sizes and complexities. Each dataset represents a binary classification problem. Table 1 outlines the complexity of each dataset in terms of number of instances and number of features prior to preprocessing.

Each dataset was sent through basic preprocessing in preparation for modeling. Duplicate rows and columns with greater than 50% missingness were dropped. Missing values were imputed using the median value for numeric columns and with the mode for categorical columns. Numeric features were standardized, and categorical variables were encoded based upon their cardinality. Categorical features with only two unique values were encoded as binary flags. Categorical features with three to ten unique values were one-hot encoded, creating one column per value of the feature. If a categorical feature contained more than ten unique values, the feature was target encoded, replacing each value with the averaged class prediction of all instances with that value [39]. Not all models handle sparsity well, so this encoding scheme was chosen to prevent extreme increases in data dimensionality. 80% of the data was set aside for training and the remaining 20% was held for testing.

D. Model Training and Tuning

Five models were selected for testing: Logistic Regression (LOGREG), Decision Trees (DT), Random Forest (RF), XGBoost (XGB), and Support Vector Machines (SVC). These models were chosen for their popularity in traditional machine learning and to test the effectiveness of SHAP selection across models with varying degrees of built-in feature selection. Since SHAP values are used to provide explanations for trained model predictions, they are able to leverage the benefits of embedded feature selection that is already present in the modeling algorithms. Thus, SHAP is able to piggyback on the feature selection already performed internally within each model and is sensitive to hyperparameter tuning.

Each model was trained with an exhaustive search of a small to moderate sized hyperparameter grid. Tuning and scoring

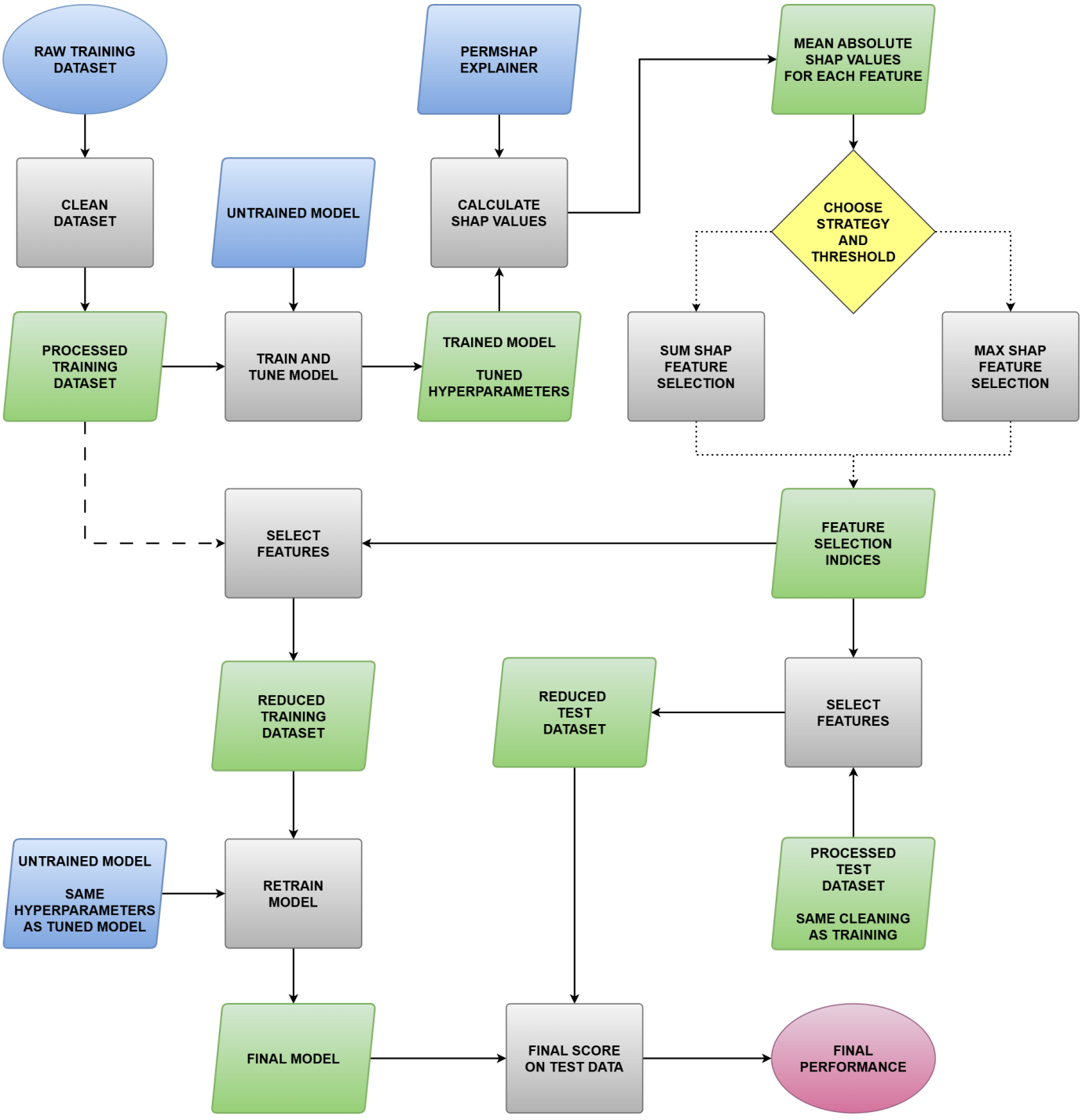

Fig. 1. Project architecture for SHAP selection process. Blue nodes indicate inputs, green nodes indicate derived inputs/outputs, gray nodes indicate actions and processes, yellow nodes indicate decisions, and the pink circle indicates the final performance output. Dotted lines indicate a split path, while dashed lines are used for reused inputs. This process is repeated for each of the MAX and SUM strategies, each of the six thresholds for each of the strategies, and for each of the five model types for each of the ten datasets.

was performed using 5-fold cross-validation. Area Under the Precision-Recall Curve (PR AUC) was chosen as the metric to optimize, as a higher PR AUC indicates a more robust model and is more informative, stable, and relevant for datasets with large class imbalances than other common classification metrics, such as Area Under the Receiver Operator Characteristic (ROC AUC) or F1 score [40]. A list of the final hyperparameters used for each dataset-model combination is given in the Appendix.

E. Performance Evaluation

Three filter-based methods were chosen for comparison with our SHAP selection algorithms: Mutual Information Gain (MI), ReliefF, and mRMR. Each of these algorithms simply ranks the features and requires the user to select the resulting k features they wish to use for selection. Each of these filter methods has their own set of hyperparameters, as discussed in the Appendix. In order to provide useful comparisons, the MAX and SUM SHAP selection strategies were employed to generate sets of candidate k values for each dataset, and these sets of k values were used to test MI, ReliefF, and mRMR, respectively.

For each dataset, each model was trained using the full dataset, and hyperparameters were selected. The same models with fixed hyperparameters were then retrained on all candidate feature subsets as discovered by the SUM and MAX SHAP selection processes and all candidate feature subsets as discovered by MI, ReliefF, and mRMR. Model performance was scored using 5-fold cross-validation with PR AUC as the primary metric.

The experimentation process was repeated for each model type, feature selection technique, and dataset. Since there were six thresholds used for each of the MAX and SUM SHAP selection strategies, a total possible 12 candidate k values were chosen per dataset-model combination. Four feature selection techniques, SHAP, MI, ReliefF, and mRMR, were tested, resulting in roughly 48 reduced feature sets created per dataset-model combination. The MAX and SHAP strategies are influenced by model type, so this process was repeated for each of the five model types tested. The end result was that approximately 241 feature sets were tested for each dataset: the full feature set plus 240 reduced feature sets generated by 12 candidate k values times four selection techniques times five model types. Across all ten datasets, almost 2,410 experiments were performed. This number is approximate as not all dataset-model combinations produced 12 unique candidate k values, since sometimes the MAX and SUM strategies chose the same feature subsets.

The feature selection methods were compared across four categories: reduction in model overfitting as measured by difference in cross-validation and test scores, overall test performance, the degree of feature reduction achieved, and the computational efficiency of the algorithm. An ideal feature selection technique can achieve high test performance and lower the amount of model overfitting while removing as many features and taking as little time as possible. In practice, no one feature selection technique can be the best at all categories. As such, it is necessary to compare relative performance gain or loss compared to feature reduction attained and computational efficiency to weigh the benefits and drawbacks of each technique.

Our project utilized the standard Python machine learning libraries Scikit-learn, Pandas and NumPy, along with the shap [41] package for generating SHAP explanations, the ReliefF library [42] for relief-based feature selection, and the scikit-fda library [43] for mRMR selection. All of the files, experiments, and code can be found in our GitHub repository at [44].

V. Analysis

A. Frameworks for Choosing the "Best" Value for k

Deciding on the best number of features to keep is not a simple task. There are multiple approaches for choosing the optimal feature subset, depending on the desired outcome. We have decided to evaluate our SHAP feature selection strategies using four different goals: minimization of overfitting, maximization of test performance, strongest peak test performance, and earliest peak test performance. A description of these four goals follows. Remember that the main metric for model performance is PR AUC.

The minimum overfitting goal focuses on the elimination of noisy and sparse patterns within the data. Suppose the cross-validation performance when using k features is $s_{cv}(k)$ and the test performance is $s_{test}(k)$. Then, the optimal feature choice for k^* is the feature subset such that the difference in cross-validation and test performance is minimized:

$$k^* = \min_k \left(\arg\min_k \left(s_{cv}(k) - s_{test}(k) \right) \right) \tag{6}$$

We choose the minimum k value in case there are ties for $\arg\min_k$. Note that this difference can be negative, as the model may perform better on the test set than on the cross-validation set. Typically, we have found that the k^* which minimizes overfitting also keeps very few features, since overfitting tends to be reduced by removing features, sometimes leading to model underfitting due to lack of information.

The remaining three frameworks for choosing the best selection for k are illustrated in Fig. 2. The maximum test score goal focuses purely on maximized performance. In other words, under the maximum test score framework, k^* is given by:

$$k^* = \min_k \left(\arg\max_k \ s_{test}(k) \right) \tag{7}$$

Our experimentation revealed that this choice of k^* tends to result in the least amount of feature reduction.

The final two frameworks rely on the concept of "peaks" in the test scores. A peak is a local maximum among the test scores. To avoid choosing noisy peaks created due to underfitting, we have removed any peaks whose test scores

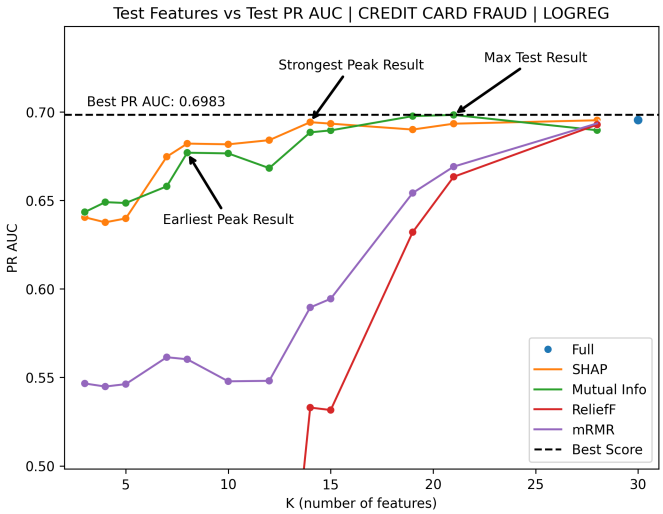

Fig. 2. Annotated example of three of the four frameworks for choosing optimal k values for each feature selection technique. The maximum test k is chosen when the test metric is maximized. Peaks occur when the test performance hits a local maximum. The earliest peak is the first peak in test performance, which corresponds to a greedy forward-selection technique. The strongest peak is the peak with the highest test performance. SHAP typically outperforms other techniques, especially when the number of features kept is lower and higher feature reduction is achieved.

are below the 0.33 quantile for test scores. Mathematically, the set of local maxima $k^{\#}$ are defined as:

$$k^{\#} = \Big\{ k \;\mid\; s_{test}(k-1) < s_{test}(k) < s_{test}(k+1) \;\wedge\; s_{test}(k) \geq Q_{0.33}(s_{test}) \Big\} \quad (8)$$

where Q is the quantile function. The peaks in $k^{\#}$ do not include the lowest or highest k values unless no peaks are found, which only occurs when the test score function is monotonic. Therefore, the strongest peak k^* is given by:

$$k^* = \min_k \left(\{ k \in k^{\#} \mid \arg\max_k \; s_{test}(k) \} \right) \quad (9)$$

Thus, the strongest peak is simply the k in $k^{\#}$ with the maximal test performance. The earliest peak k^* is just the lowest k in $k^{\#}$ given by:

$$k^* = \min_k \{ k \in k^{\#} \} \quad (10)$$

From our experiments, we have found that the frameworks generally produce optimal k^* values in the following order from lowest k to highest k: minimum overfitting, earliest peak, strongest peak, maximum test. There is no guarantee that this holds true for all datasets, but it is a common trend in our experiment results. Performance on the test set tends to increase as k increases, but overfitting tends to decrease as k decreases, meaning the optimal value for k is dependent upon the user's preference for balancing performance with feature reduction. Fig. 3 summarizes the number of dataset-model combinations where each selection type outperformed the other selection types in test performance, along with the average percentage of features kept by each strategy among these "winning" dataset-model counts.

Counts and Mean Features Kept by Framework, Global Counts

Selection Type	Min Overfit Count	Min Overfit Mean K%	Max Test Count	Max Test Mean K%	Strongest Peak Count	Strongest Peak Mean K%	Earliest Peak Count	Earliest Peak Mean K%
mRMR	5	21.36	5	75.76	7	74.85	4	72.46
Mutual Information	9	37.51	10	54.59	11	54.05	12	46.28
ReliefF	14	20.18	2	90.00	0	0.00	2	49.85
SHAP	10	16.42	24	46.21	25	43.60	26	39.38

Fig. 3. Counts and mean features kept for each feature selection technique with respect to each framework. A technique was counted as the best for the dataset-model-framework combination if the technique exhibited the best performance with the least features. If two selection types were tied for performance and features kept in the dataset-model-framework combination, both counts were omitted. In frameworks where a technique was never selected, the count and average percentage of features kept was set to 0.

B. SHAP Selection Versus Filter Methods Across Frameworks

Each of the selection methods was evaluated under each of the aforementioned frameworks. Fig. 3 shows the counts of dataset-model combinations where each selection technique had the best performance, where best performance was defined as the lowest percentage difference between cross-validation PR AUC and test PR AUC under the minimum overfitting framework and defined as the highest test PR AUC under all other frameworks.

For three of the four frameworks, SHAP outperformed each of the filter-based selection methods. SHAP performed best when optimizing for maximum test performance, strongest test peak, and earliest test peak. When optimizing for minimum overfitting, ReliefF outperformed SHAP; however, it is worth noting that ReliefF consistently had worse performance than all other feature selection techniques for all dataset-model combinations. This can be seen in Fig. 2, where the ReliefF score curve is lower than the score curves for all other selection techniques. In this way, ReliefF minimized overfitting by causing the models to underfit. For most dataset-model combinations, MI and SHAP provided similar performance, although SHAP usually selected fewer features for similar performance. mRMR often produced better results than ReliefF, but also usually marginally or significantly worse results than MI and SHAP.

For all frameworks, SHAP consistently achieved equivalent or greater performance when compared to the filter methods while choosing fewer features. This indicates that using SHAP explanations produces high quality feature rankings, as the strongest features from SHAP outperform the strongest features from the filter methods. SHAP is able to obtain much greater feature reduction by leveraging the patterns learned during modeling, giving SHAP an advantage in ranking features.

Fig. 4 shows a more comprehensive breakdown of performance counts by model type. SHAP was the most effective technique when working with decision trees. Decision trees make their splits based on information gain, choosing the strongest features which provide the purest splits. SHAP is

able to capture these patterns when ranking the features, using the tree information itself to select the best features for producing optimal splits. Since none of the filter techniques can leverage the knowledge learned from the decision tree, they tend to keep suboptimal feature sets for the tree models. SHAP was also the most commonly chosen best technique for the logistic regression model, being able to use the built-in L_1 or L_2 regularization to eliminate unimportant features. Notably, SHAP tends to keep slightly more features on average than MI for logistic regression models. This may be due to logistic regression coefficients being sensitive to multicollinearity, and since the SHAP values are calculated on the model trained on the full dataset with no prior feature selection, there is a higher chance for multicollinearity to affect SHAP feature rankings, even when leveraging embedded regularization techniques.

Although MI is the most commonly chosen best technique for random forest models, SHAP selects significantly fewer features. As with the decision trees, random forests make splits based on information gain, so SHAP is able to identify the most informative features discovered by the forest. This logic does not appear to apply to XGBoost models, where MI often chose fewer features than SHAP. This may be because XGBoost is a gradient boosting tree ensemble, so each individual learner only sees a subset of features and learns weakly. Each tree may have different feature rankings depending on its order in the boosting process, leading to less pronounced feature importances for SHAP to identify.

mRMR typically underperformed in comparison to MI or SHAP for most dataset-model combinations; however, it excelled when used in conjunction with support vector machine models. SVC models are susceptible to collinearity, especially when using a linear kernel, and the mRMR algorithm naturally helps remove collinearity by reducing redundancy. The SHAP values are calculated using the SVC trained on the full feature set, and so the rankings are somewhat affected by feature collinearity. The SVC performance is more robust to collinearity when using nonlinear kernels but becomes more computationally complex, so these nonlinear kernels were only used for the smaller and less complex datasets.

C. MAX and SUM Strategy Comparisons

Thus far, we have already shown how SHAP provides superior or equivalent model performance with greater feature reduction. However, we have yet to answer which of the MAX or SUM strategies is more useful, as well as which thresholds were the most effective. Fig. 5 shows the counts for the MAX and SUM thresholds under each selection framework. The most performant strategy was the MAX strategy using a 1% threshold, being the best strategy-threshold combination for maximizing test performance, finding the strongest peak performance, and when finding the earliest peak performance. The second most performant strategy-threshold combination was the MAX strategy with a 5% threshold. The MAX strategy with a 50% threshold best minimized overfitting compared to any other strategy and threshold. Surprisingly, the SUM strategy appeared to underperform, selecting suboptimal values

Counts and Mean Features Kept by Framework, Model-Specific Counts

Model	Selection Type	Min Overfit Count	Min Overfit Mean K%	Max Test Count	Max Test Mean K%	Strongest Peak Count	Strongest Peak Mean K%	Earliest Peak Count	Earliest Peak Mean K%
Decision Tree	mRMR	2	6.39	0	0.00	0	0.00	0	0.00
	Mutual Information	0	0.00	2	32.58	1	31.82	2	46.34
	ReliefF	1	3.33	0	0.00	0	0.00	0	0.00
	SHAP	3	11.39	7	20.74	8	19.39	7	15.64
Logistic Regression	mRMR	1	50.00	1	86.36	1	86.36	0	0.00
	Mutual Information	3	18.69	3	53.33	3	53.33	3	40.00
	ReliefF	4	23.24	1	88.00	0	0.00	0	0.00
	SHAP	2	17.00	5	51.41	6	54.17	7	45.56
Random Forest	mRMR	1	4.00	0	0.00	2	72.58	2	87.65
	Mutual Information	2	65.00	3	75.80	4	52.35	2	41.00
	ReliefF	2	16.61	1	92.00	0	0.00	1	63.33
	SHAP	3	26.55	3	43.06	1	33.72	3	39.64
Support Vector Machine	mRMR	1	40.00	3	73.23	3	73.23	1	60.00
	Mutual Information	1	50.00	0	0.00	1	100.00	2	71.67
	ReliefF	5	27.98	0	0.00	0	0.00	1	36.36
	SHAP	1	8.33	4	76.28	4	76.28	4	67.28
XGBoost	mRMR	0	0.00	1	72.73	1	72.73	1	54.55
	Mutual Information	3	33.84	2	46.67	2	46.67	3	39.11
	ReliefF	2	6.57	0	0.00	0	0.00	0	0.00
	SHAP	1	8.00	5	54.52	6	45.14	5	41.48

Fig. 4. Performance counts for each feature selection technique by model type. SHAP performs particularly well for Decision Tree, Logistic Regression, and XGBoost models. Although MI outperforms SHAP by scores for Random Forest models, SHAP selects far fewer features. Support Vector Machines were the only model which regularly performed better using mRMR. ReliefF underperformed for all model types.

SHAP MAX and SUM Threshold Counts by Framework

Selection Type	Min Overfit Count	Max Test Count	Strongest Peak Count	Earliest Peak Count
Max = 0.01	2	15	10	10
Max = 0.05	4	10	8	3
Max = 0.1	7	6	6	5
Max = 0.15	3	2	5	7
Max = 0.25	4	1	1	2
Max = 0.5	14	4	3	3
Sum = 0.5	6	0	0	0
Sum = 0.6	4	0	1	4
Sum = 0.7	1	2	2	2
Sum = 0.8	2	4	7	8
Sum = 0.9	1	1	2	1
Sum = 0.95	2	5	5	5

Fig. 5. Most common threshold selections for the MAX and SUM algorithms across selection frameworks. The MAX strategy was chosen more frequently than the SUM strategy.

for k in more scenarios than the MAX strategy. This result may be puzzling at first, but the causes for this phenomenon are more clear when investigating the number of features kept at different thresholds for each strategy.

Consider the MAX threshold plot in Fig. 6. As the threshold ρ increases, the number of features selected decreases until reaching a plateau. This allows the MAX algorithms to select

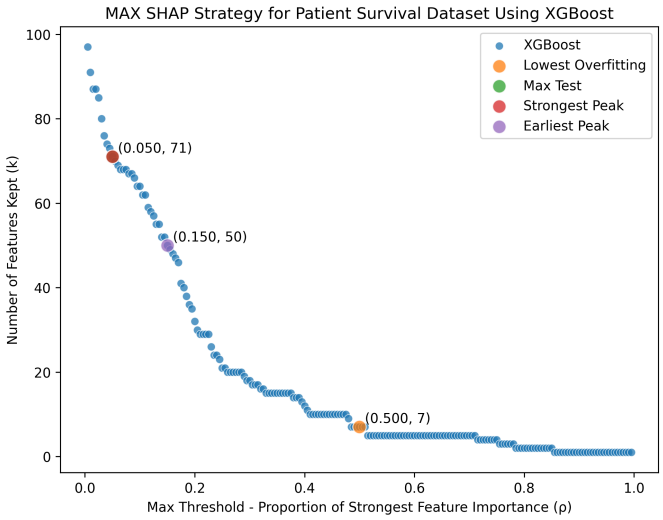

Fig. 6. Number of features kept at varying MAX thresholds for the XGBoost model on the Patient Survival dataset. Notice how the maximum test and strongest peak performance lies near a "corner" where the number of features kept flattens for a moderate change in ρ. Similarly, the lowest overfitting performance occurs near long plateaus.

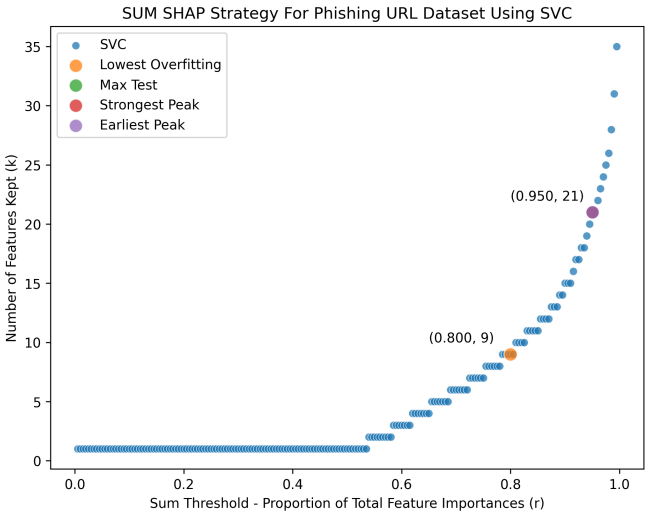

Fig. 7. Number of features kept at varying SUM thresholds for the SVC model on the Phishing URL dataset. New features are added only if the sum of the prior feature importances is insufficient, resulting in long plateaus. This leads to a less smooth choice of k compared to the MAX strategy. At high thresholds, many weak features are added at once to cover the required summed contributions, degrading performance.

features with granularity. Although we purposefully tested non-consecutive ρ values, we recommend that for using the MAX strategy, the user views this MAX threshold plot to make their decision. If the number of features kept drops swiftly between consecutive ρ values, then there are many weaker features that are discarded for a small increase in ρ. For example, between $\rho = 0$ and $\rho \approx 0.09$ in Fig. 6, about 20% of the features are dropped. Although it is impossible to verify without testing the model performance at each intermediate threshold, we hypothesize that the best candidate choices for ρ are located where the threshold curve becomes relatively flat, even if temporarily.

Now, consider the SUM threshold plot in Fig. 7. Unlike the MAX threshold plot, the SUM threshold plot consists almost entirely of long plateaus followed by sudden jumps in k. This is because a new feature is only added if the sum of all previous features importances is too low. When the current set of features is not sufficient, a new feature is added. This makes the addition of features using the SUM strategy more stochastic. In addition, notice how as r approaches 1, the length of each plateau shortens. This is because each consecutive feature that is added is weaker and contributes less to the summed importances, so at high choices of r, many weak features are added, reducing the effectiveness of the SUM strategy. We hypothesize that the SUM strategy can still be used by looking for threshold ranges where plateaus occur and by avoiding threshold ranges where each plateau is short.

We tested the MAX and SUM algorithms separately for this paper. However, it could be possible to combine them by defining both ρ and r and taking the intersection of features selected by both algorithms. Since both algorithms use the sorted feature strengths, this is equivalent to taking the minimum of the k values chosen. Alternatively, to fix the issue where the SUM algorithm begins to accumulate weak features for high values of r, the SUM algorithm could be modified such that the user specifies a constant ω which is a lower-bound proportion ratio on the total sum of importances. Any features with importance below $\omega \cdot S$ would be rejected. This effectively puts a cap on the choice of k for the SUM strategy, as higher r thresholds would fail to add more features once all features with importance greater than $\omega \cdot S$ are exhausted.

D. Computational Complexity

The time performances for each feature selection type were recorded and averaged across all dataset-model combinations for comparison in Fig. 8. MI required the least amount of time of all tested techniques and scaled exceptionally with the number of instances and features. ReliefF uses a nearest-neighbors algorithm to compute feature rankings, and as such, scales reasonably well as the number of features grows but poorly with the number of instances. mRMR calculates the mutual information between each feature and the target and the mutual information between each new candidate feature and the currently selected feature set. Because of its iterative nature requiring pairwise comparisons between features, mRMR scales poorly with the number of features but scales reasonably with the number of instances. SHAP values need to be computed for every instance in order to compute global averaged feature importances, and the SHAP values require searching through the feature space, as outlined in Section II, to calculate the relative contribution of each feature away from the mean prediction. This means that SHAP scales with both the number of instances and the number of features, resulting in the worst complexity of all of the tested techniques. SHAP was more than 10 times slower than the other selection techniques on average.

Time Performance Comparisons	
Selection Type	**Average Time (s)**
mRMR	972.58
Mutual Information	17.43
ReliefF	495.78
SHAP	12,606.50

Fig. 8. Average time in seconds required to select features. Note that the mRMR algorithm is overestimated, as the mRMR algorithm is iterative and was run once for the full feature set and then sliced to prevent having to recalculate mRMR for each k value.

Although SHAP provides state-of-the-art test performance with exemplary feature reduction, a drawback to SHAP values is that they are computationally expensive. The PermSHAP algorithm used in this paper is model-agnostic but is not optimized for any model type. This makes the calculation of SHAP values time-consuming, especially for ensemble methods such as Random Forest or XGBoost which require inference across many learners, as shown in Fig. 9. It is important to note that the TreeSHAP algorithm described earlier [21] is optimized for tree-based algorithms and would produce extreme improvements for the tree-based models. SHAP required between 8 and 55 times the amount of time to calculate for Random Forest models than for simpler models, keenly highlighting the reason most prior work using SHAP for feature selection utilized the TreeSHAP algorithm.

A reasonable concern is whether SHAP is worth the additional time complexity. MI produced similar, or sometimes superior, results to SHAP for most of our tested dataset-model combinations. On average, MI completed its feature selection in 0.13% of the time required to calculate SHAP values. SHAP consistently produced greater feature reduction, but at greater computational cost. If a user prefers performance and speed and is less concerned with creating the most pruned feature set, then based upon our experiments, we would recommend using MI. However, if time and compute power is not a concern and the goal is to reduce the feature set as much as possible, SHAP selection would be preferable.

VI. Conclusion and Future Steps

Our experiments show that SHAP is a highly effective feature selection technique. While originally created for the purpose of machine learning explainability, SHAP values can be repurposed into an accurate method for feature ranking. Because SHAP treats the model as a black box, SHAP can be used as a model-agnostic feature selection technique that is able to leverage the patterns and knowledge discovered during model training to choose high performance features, seamlessly combining with any built-in embedded feature selection. SHAP captures feature subsets which exhibit equivalent or superior performance to feature subsets selected by common filter-based feature selection techniques while resulting in greater feature reduction.

SHAP Computation Times (seconds)					
	Decision Tree	**Logistic Regression**	**Random Forest**	**SVC**	**XGBoost**
Android Permissions	1,025.19	1,000.07	32,670.93	947.05	3,675.71
Breast Cancer	9.33	14.85	71.40	92.84	42.37
Credit Card Fraud	3,228.36	2,888.66	127,716.21	3,736.95	40,105.31
Heart Disease	2.96	8.21	28.71	55.64	8.26
Indian Liver	4.91	9.94	210.89	81.55	14.76
Mushroom	243.64	229.65	871.69	166.10	610.68
Patient Survival	5,764.92	6,955.44	187,604.88	3,362.46	30,743.15
Phishing URL	4,164.31	4,002.62	110,062.90	3,998.73	17,241.88
SPECT Heart	4.63	0.93	22.07	9.93	9.41
Secondary Mushroom	1,305.18	1,200.42	24,108.15	958.08	9,032.26

Fig. 9. Permutation SHAP calculation times for each model type and dataset. The Permutation SHAP explainer is model-agnostic but scales poorly with model complexity, dataset size, and number of features. The computation times for Support Vector Machines is lower on the larger datasets due to the switch from the RBF kernel to the faster linear kernel.

The MAX and SUM strategies created for this experiment proved effective for choosing the correct number of features to retain in a way that is simple, intuitive, and forgoes expensive iterative processes. In particular, the MAX strategy provides a natural way to filter out weak features by enforcing feature importances to be relevant with respect to the scale of the strongest feature importance. Our experiments showed that the MAX strategy performed well for most datasets using thresholds between 1% and 10% of the maximum feature strength, and we provided a potential strategy for using the MAX threshold curve for manually selecting optimal thresholds.

Despite the performance benefits of using SHAP as a feature selection technique, our experiments have shown that SHAP is a computationally expensive technique. SHAP scales poorly with dataset size and the size of the feature space, especially when using the model-agnostic Permutation explainer. Time and resources permitting, SHAP produces superior results to the filter methods that were tested, but SHAP may not be worthwhile to implement for sufficiently large and complex datasets.

We view three main paths forward to build upon the work in this paper. The first is to modify the SUM strategy to filter out weak features at higher thresholds. Adding weak features simply to meet the desired summed importance threshold is counterproductive to the purpose of feature selection.

The second path is to expand the MAX and SUM algorithms to operate in automatic or semi-automatic settings. Under the current implementation, the user must specify the threshold used. While this ensures the chosen threshold aligns with

Model Hyperparamaters per Dataset					
Model Type	**Decision Tree**	**Logistic Regression**	**Random Forest**	**Support Vector Machine**	**XGBoost**
Dataset					
Android Permissions	max_depth = None min_samples_split = 2 min_samples_leaf = 20	solver = saga penalty = l2 C = 10	n_estimators = 600 max_depth = None min_samples_split = 2 min_samples_leaf = 1	kernel = linear C = 0.1	n_estimators = 200 max_depth = 5 min_child_weight = 1 subsample = 0.9 colsample_bytree = 0.7
Breast Cancer	max_depth = 5 min_samples_split = 2 min_samples_leaf = 10	solver = saga penalty = l1 C = 10	n_estimators = 100 max_depth = 5 min_samples_split = 2 min_samples_leaf = 1	kernel = rbf C = 10 gamma = 0.01	n_estimators = 400 max_depth = 5 min_child_weight = 1 subsample = 0.7 colsample_bytree = 0.8
Credit Card Fraud	max_depth = 9 min_samples_split = 2 min_samples_leaf = 10	solver = saga penalty = l1 C = 100	n_estimators = 400 max_depth = None min_samples_split = 2 min_samples_leaf = 1	kernel = linear C = 10	n_estimators = 400 max_depth = 4 min_child_weight = 5 subsample = 0.8 colsample_bytree = 0.8
Heart Disease	max_depth = 3 min_samples_split = 2 min_samples_leaf = 10	solver = saga penalty = l2 C = 0.1	n_estimators = 100 max_depth = 5 min_samples_split = 6 min_samples_leaf = 1	kernel = rbf C = 1 gamma = 0.01	n_estimators = 100 max_depth = 3 min_child_weight = 5 subsample = 0.7 colsample_bytree = 0.8
Indian Liver	max_depth = None min_samples_split = 2 min_samples_leaf = 20	solver = saga penalty = l2 C = 1	n_estimators = 600 max_depth = 5 min_samples_split = 2 min_samples_leaf = 10	kernel = rbf C = 100 gamma = 0.01	n_estimators = 100 max_depth = 4 min_child_weight = 1 subsample = 1 colsample_bytree = 0.8
Mushroom	max_depth = None min_samples_split = 2 min_samples_leaf = 10	solver = saga penalty = l1 C = 1	n_estimators = 100 max_depth = None min_samples_split = 2 min_samples_leaf = 1	kernel = linear C = 1000	n_estimators = 100 max_depth = 3 min_child_weight = 1 subsample = 0.7 colsample_bytree = 0.7
Patient Survival	max_depth = 7 min_samples_split = 2 min_samples_leaf = 20	solver = saga penalty = l1 C = 1	n_estimators = 400 max_depth = None min_samples_split = 6 min_samples_leaf = 1	kernel = linear C = 1000	n_estimators = 400 max_depth = 3 min_child_weight = 3 subsample = 1 colsample_bytree = 1
Phishing URL	max_depth = None min_samples_split = 2 min_samples_leaf = 1	solver = lbfgs penalty = l2 C = 0.1	n_estimators = 400 max_depth = None min_samples_split = 2 min_samples_leaf = 1	kernel = linear C = 0.01	n_estimators = 400 max_depth = 4 min_child_weight = 1 subsample = 1 colsample_bytree = 0.8
SPECT Heart	max_depth = None min_samples_split = 2 min_samples_leaf = 1	solver = saga penalty = l2 C = 100.0	n_estimators = 100 max_depth = 5 min_samples_split = 10 min_samples_leaf = 1	kernel = rbf C = 100 gamma = 0.1	n_estimators = 100 max_depth = 3 min_child_weight = 3 subsample = 0.7 colsample_bytree = 0.7
Secondary Mushroom	max_depth = None min_samples_split = 2 min_samples_leaf = 10	solver = saga penalty = l1 C = 1	n_estimators = 200 max_depth = None min_samples_split = 2 min_samples_leaf = 1	kernel = linear C = 1000	n_estimators = 400 max_depth = 5 min_child_weight = 1 subsample = 0.7 colsample_bytree = 0.7

Fig. 10. Model hyperparameters for each dataset. Each model was hyperparameter tuned using 5-fold cross-validation with PR AUC as the scoring metric. Each model was trained with a fixed random state of 42 for reproducibility. Datasets with high complexity were trained using reduced hyperparameter grids and faster algorithms as needed, due to computing power limits. Examples include linear kernels for SVC or the lbfgs solver for Logistic Regression.

human intuition, this implementation invites human error. It would be worthwhile to expand the algorithms in order to analyze the MAX/SUM threshold curves to suggest promising candidate thresholds.

The third path forward would be to explore improving the efficiency of collecting SHAP values. This paper utilized the model-agnostic Permutation explainer, but it is possible to experiment with SHAP explainers that are optimized for the selected model types. In addition, it may be possible to estimate the global SHAP feature importances without having to calculate SHAP values for every sample. If the SHAP feature importances can be approximated to an acceptable degree without using all samples, then the time complexity of the SHAP algorithms would not grow with the size of the dataset, or the time complexity might grow more slowly.

APPENDIX

While it is possible to perform these experiments without hyperparameter tuning, one of the main benefits of SHAP explanations is the ability to uncover patterns discovered by models, including innate feature selection provided by embedded methods such as max depth pruning in tree-based models or regularization techniques in logistic regression and support vector machines.

The model hyperparameters chosen for each model and dataset are outlined in Fig. 10. Each model was trained using a moderately-sized hyperparameter grid, ranging from 10 combinations for simple models, such as logistic regression, to 432 combinations for complex models, such as XGBoost. An exhaustive grid search was performed using 5-fold cross-validation to select optimal hyperparameters.

For large or high-dimensionality datasets, the hyperparameter grids were restricted to reduce the search space and make the training computationally feasible. The main offending model types for difficult datasets were the random forest and XGBoost models, where the size of the grids were reduced from 180 combinations to 6 combinations and from 432 combinations to 36 combinations, respectively. The logistic regression models sometimes struggled for larger datasets when using the saga solver which supports both L_1 and L_2 regularization, so in these situations, the solver was changed to the lbfgs algorithm which only supports L_2 regularization. Support vector machines are highly flexible when using radial-basis-function (rbf) kernels, but the kernel transformations are extremely costly for high dimensionality datasets. For these datasets, the linear kernel was used instead.

Each of the feature selection techniques featured in this paper have hyperparameters that can be modified. For the PermSHAP explainer, there are two main hyperparameters required [20]. The first is the background dataset used to mask out hidden features. The Permutation explainer masks features while iterating through permutations, drawing masked samples from the background data. By using different background data, different masked samples are drawn, resulting in different SHAP values. For our experiments, the full training dataset was always passed as the background data. The second hyperparameter is the number of times to cycle through all of the features during the permutation process. Higher numbers of cycles grants better SHAP estimates by capturing higher order interactions. By default, PermSHAP performs 10 cycles, which is what we used for our experiments. However, it should be noted that the Permutation explainer needs to evaluate the model function 2 times (number of features + 1) times (number of background data samples) times (number of cycles), meaning that the calculation of SHAP explanations grows quadratically with the number of samples. If n is the number of samples, p is the number of features, and c is the number of cycles, then the number of model evaluations by PermSHAP using our selected hyperparameters per dataset is

$$n(2cn(p+1)) \rightarrow O(cpn^2) \tag{11}$$

with respect to model complexity, which explains the poor time complexity of SHAP explanations in our experiments.

The mutual information implementation in scikit-learn uses a nearest neighbors algorithm for estimating entropy for continuous features. Scikit-learn uses a default of 3 neighbors, which we used for our experiments. Higher numbers of neighbors reduce variance but can introduce a bias, and we saw no reason to modify this hyperparameter. ReliefF is a distance-based algorithm and also uses a nearest neighbors parameter, where 2x neighbors are evaluated, once for instances belonging to the same class and once for instances belonging to the opposite class. The default number of neighbors for the package we used is 100 [42]; however, we found this to be computationally infeasible for the larger datasets and to be impossible for smaller datasets due to their lack of samples. To make computation feasible, we reduced the number of neighbors for ReliefF down to 3 neighbors, which reduces the accuracy of the scores and may explain the poor performance of the ReliefF algorithm for the majority of dataset-model combinations. For the mRMR algorithm, the main hyperparameter is method chosen. The mRMR adds features iteratively based on the chosen method. The common choices for methods with a classification target variable are the mutual information difference (MID) and mutual information quotient (MIQ). As suggested by their names, MID is the difference in MI of a feature with respect to the target variable and the MI of a feature with respect to the currently selected feature set. MIQ is the same but instead of the difference, the ratio of MIs is used. MID is the method used in [10], so that is the method we chose to use for our experiments.

REFERENCES

[1] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," Journal of Machine Learning Research, vol. 3, pp. 1157–1182, 2003.

[2] H. Abdi and L. J. Williams, "Principal component analysis," WIREs Comput. Stat., vol. 2, no. 4, pp. 433–459, 2010, doi: 10.1002/wics.101

[3] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," Journal of Machine Learning Research, vol. 9, pp. 2579–2605, 2008.

[4] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," arXiv preprint, arXiv:1802.03426, 2018.

[5] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," Computers & Electrical Engineering, vol. 40, no. 1, pp. 16–28, 2014, 40th-year commemorative issue.

[6] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in Proc. 38th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO), May 2015, pp. 1200–1205.

[7] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[8] W. E. Marcílio and D. M. Eler, "From explanations to feature selection: Assessing SHAP values as feature selection mechanism," in Proc. 33rd SIBGRAPI Conf. Graph., Patterns Images (SIBGRAPI), Porto de Galinhas, Brazil, 2020, pp. 340–347, doi: 10.1109/SIBGRAPI51738.2020.00053.

[9] Y. B. Wah, N. Ibrahim, H. A. Hamid, S. Abdul-Rahman, and S. Fong, "Feature selection methods: Case of filter and wrapper approaches for maximising classification accuracy," Pertanika J. Sci. Technol., vol. 26, no. 1, 2018.

[10] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," IEEE Trans. Pattern Anal. Mach. Intell., vol. 27, no. 8, pp. 1226–1238, Aug. 2005, doi: 10.1109/TPAMI.2005.159.

[11] X. Lin, C. Li, W. Ren, X. Luo, and Y. Qi, "A new feature selection method based on symmetrical uncertainty and interaction gain," Computational Biology and Chemistry, vol. 83, p. 107149, 2019, doi: 10.1016/j.compbiolchem.2019.107149.

[12] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," J. Biomed. Inform., vol. 85, pp. 189–203, 2018, doi: 10.1016/j.jbi.2018.07.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046418301400

[13] Q. Gu, Z. Li, and J. Han, "Generalized Fisher score for feature selection," arXiv preprint, arXiv:1202.3725, 2012. [Online]. Available: https://arxiv.org/pdf/1202.3725

[14] B. Xue, M. Zhang, and W. N. Browne, "A comprehensive comparison on evolutionary feature selection approaches to classification," Int. J. Comput. Intell. Appl., vol. 14, no. 2, p. 1550008, 2015.

[15] N. El Aboudi and L. Benhlima, "Review on wrapper feature selection approaches," in Proc. 2016 Int. Conf. Eng. MIS (ICEMIS), Sept. 2016, pp. 1–5.

[16] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in Proc. 2014 Sci. Inf. Conf., Aug. 2014, pp. 372–378.

[17] C. Molnar, Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 3rd ed., 2025. [Online]. Available: https://christophm.github.io/interpretable-ml-book/

[18] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable AI: A review of machine learning interpretability methods," Entropy, vol. 23, no. 1, p. 18, 2020.

[19] R. Mitchell, J. Cooper, E. Frank, and G. Holmes, "Sampling permutations for Shapley value estimation," J. Mach. Learn. Res., vol. 23, no. 43, pp. 1–46, 2022.

[20] SHAP Documentation – PermutationExplainer API, [Online]. Available: https://shap.readthedocs.io/en/latest/generated/shap.PermutationExplainer.html

[21] S. M. Lundberg, G. G. Erion, and S. I. Lee, "Consistent individualized feature attribution for tree ensembles," arXiv preprint, arXiv:1802.03888, 2018. [Online]. Available: https://arxiv.org/abs/1802.03888

[22] Y. Gebreyesus, D. Dalton, S. Nixon, D. De Chiara, and M. Chinnici, "Machine learning for data center optimizations: Feature selection using Shapley additive explanation (SHAP)," Future Internet, vol. 15, no. 3, p. 88, 2023, doi: 10.3390/fi15030088.

[23] E. Kraev, B. Koseoglu, L. Traverso, and M. Topiwalla, "Shap-Select: Lightweight feature selection using SHAP values and regression," arXiv preprint, arXiv:2410.06815, 2024, doi: 10.48550/arXiv.2410.06815.

[24] H. Wang, Q. Liang, J. T. Hancock, et al., "Feature selection strategies: A comparative analysis of SHAP-value and importance-based methods," J. Big Data, vol. 11, no. 44, 2024, doi: 10.1186/s40537-024-00905-w.

[25] M. S. H. Shaon, T. Karim, M. S. Shakil, and M. Z. Hasan, "A comparative study of machine learning models with LASSO and SHAP feature selection for breast cancer prediction," Healthcare Anal., vol. 6, p. 100353, 2024, doi: 10.1016/j.health.2024.100353.

[26] E. Keany, "BorutaShap: A wrapper feature selection method which combines the Boruta feature selection algorithm with Shapley values," Zenodo, 2020, doi: 10.5281/zenodo.4247610.

[27] J. Verhaeghe, J. Van Der Donckt, F. Ongenae, and S. Van Hoecke, "Powershap: A power-full Shapley feature selection method," in Mach. Learn. Knowl. Discov. Databases (ECML PKDD 2022), M. R. Amini et al., Eds. Cham: Springer, 2023, vol. 13713, pp. 49–65, doi: 10.1007/978-3-031-26387-3_5.

[28] M. B. Kursa, A. Jankowski, and W. R. Rudnicki, "Boruta–a system for feature selection," Fundam. Inform., vol. 101, no. 4, pp. 271–285, 2010.

[29] B. Ramana and N. Venkateswarlu, "ILPD (Indian Liver Patient Dataset)," UCI Mach. Learn. Repository, 2022. [Online]. Available: https://doi.org/10.24432/C5D02C

[30] A. Janosi, W. Steinbrunn, M. Pfisterer, and R. Detrano, "Heart Disease," UCI Mach. Learn. Repository, 1989. [Online]. Available: https://doi.org/10.24432/C52P4X

[31] "Mushroom," UCI Mach. Learn. Repository, 1981. [Online]. Available: https://doi.org/10.24432/C5959T

[32] K. Cios, L. Kurgan, and L. Goodenday, "SPECT Heart," UCI Mach. Learn. Repository, 2001. [Online]. Available: https://doi.org/10.24432/C5P304

[33] W. Wolberg, O. Mangasarian, N. Street, and W. Street, "Breast Cancer Wisconsin (Diagnostic)," UCI Mach. Learn. Repository, 1993. [Online]. Available: https://doi.org/10.24432/C5DW2B

[34] D. Wagner, D. Heider, and G. Hattab, "Secondary Mushroom," UCI Mach. Learn. Repository, 2021. [Online]. Available: https://doi.org/10.24432/C5FP5Q

[35] Credit Card Fraud Detection Dataset, version 1, Zenodo, Dec. 2022, doi: 10.5281/zenodo.7395559. [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[36] A. Prasad and S. Chandra, "PhiUSIIL Phishing URL (Website)," UCI Mach. Learn. Repository, 2024. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103545

[37] M. Agarwal, Patient Survival Prediction [Dataset], 2021. [Online]. Available: https://www.kaggle.com/datasets/mitishaagarwal/patient

[38] A. Mathur, "NATICUSdroid (Android Permissions)," UCI Mach. Learn. Repository, 2021. [Online]. Available: https://doi.org/10.24432/C5FS64

[39] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems," ACM SIGKDD Explor. Newsl., vol. 3, no. 1, pp. 27–32, Jul. 2001, doi: 10.1145/507533.507538

[40] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," PLOS ONE, vol. 10, no. 3, p. e0118432, 2015, doi: 10.1371/journal.pone.0118432.

[41] SHAP Documentation (Main Page), [Online]. Available: https://shap.readthedocs.io/en/latest/

[42] R. S. Olson, "ReliefF," Zenodo, 2016, doi: 10.5281/zenodo.47803.

[43] C. Ramos-Carreño, "GAA-UAM/scikit-fda: Functional Data Analysis in Python". Zenodo, Apr. 04, 2025. doi: 10.5281/zenodo.15144929.

[44] J. Gottlieb, M. Chunsen, and A. Yoo, "SHAP-Feature-Selection," GitHub Repository, accessed Nov. 2, 2025. [Online]. Available: https://github.com/JoshuaGottlieb/SHAP-Feature-Selection