EXTRACTION OF INTERPRETABLE DECISION RULES FROM BLACK-BOX MODELS FOR CLASSIFICATION TASKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EGEMEN BERK GALATALI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2022

Approval of the thesis:

EXTRACTION OF INTERPRETABLE DECISION RULES FROM BLACK-BOX MODELS FOR CLASSIFICATION TASKS

submitted by **EGEMEN BERK GALATALI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil KALIPÇILAR
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Hande Alemdar
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Tansel DÖKEROĞLU
Software Engineering, Çankaya University _____

Assoc. Prof. Dr. Hande Alemdar
Computer Engineering, METU _____

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Date: 31.08.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Egemen Berk Galatalı

Signature :

ABSTRACT

EXTRACTION OF INTERPRETABLE DECISION RULES FROM BLACK-BOX MODELS FOR CLASSIFICATION TASKS

Galatalı, Egemen Berk
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Hande Alemdar

August 2022, 51 pages

In this work, we have proposed a new method and ready to use workflow to extract simplified rule sets for a given Machine Learning (ML) model trained on a classification task. Those rules are both human readable and in the form of software code pieces thanks to the syntax of Python programming language. We have inspired from the power of Shapley Values as our source of truth to select most prominent features for our rule sets. The aim of this work to select the key interval points in given data in order to extract if-then rule sets representing the black box models. We are able to generate rules that can mimic four different ML models on three datasets and one Deep Learning model on stock price dataset. We have evaluated promising similarity scores (around 90%) between the extracted rules and the ML models.

Keywords: Interpretability, Rule Extraction, Classification, Machine Learning, Deep Learning

ÖZ

KATEGORİLENDİRME AMAÇLI KAPALI KUTU MODELLERDEN AÇIKLANABİLİR KURAL SETLERİ ÇIKARIMI

Galatalı, Egemen Berk
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Hande Alemdar

Ağustos 2022 , 51 sayfa

Bu çalışmamızda, kategorilendirme üzerine çalışan kapalı-kutu modeller için açıklanabilirlik metodu geliştirdik ve sizlere sunuyoruz. Modeli açıklamaya yönelik üretmiş olduğumuz kural setleri hem insan hem de makina tarafından okunabilir formattadır. Geliştirdiğimiz metod içerisinde Shapley değerlerinden faydalandık ve bu değerlerin önemli gördüğü aralıkları kendi kural setimizi çıkarmakta yardımcı oldular. Dört farklı makine öğrenmesi metodunu üç farklı veri seti üzerinde ve bir derin öğrenme metodunu finansal veri seti üzerinde test ettiğimizde, ümit verici sonuçlar elde ettik. Çıkarım yaptığımız kural setleri ile orijinal model arasinda %90 civarlarında benzerlik gözlemledik.

Anahtar Kelimeler: Açıklanabilirlik, kural çıkarımı, kategorilendirme, makine öğrenmesi, derin öğrenme

I would like to dedicate this work to people who have contributed to science.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Hande Alemdar for her guidance, positiveness and benevolence. I would also like to thank my friends who have supported me by uttering their trust in me throughout the process. I would like to recognize all the people in the science community for building such incredible literature to make the world a better place. I must say there were concerns in the beginning due to the unknown nature of the process. Nevertheless, I cope with that thanks to my undergrad education in the CENG department of the METU. They taught me an intuitive way to cope with never seen problems. I cannot really describe the algorithm for solving a problem. However, I can say that when I face a problem, my brain starts to seek answers. There is no fear of uncertainty. Even if the answer could not be found, I like the challenge in it and try to enjoy it while learning. About this thesis, I have really enjoyed the coding work along the journey. It was like a list of different puzzle types, each having unique solutions. Overall, it was an excellent experience to work on something I have never tried before. I would like to thank the committee members individually for their involvement.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF ABBREVIATIONS xvi

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation and Problem Definition 1

1.2 Proposed Methods and Models 1

1.3 Contributions and Novelties 2

1.4 The Outline of the Thesis 2

2 BACKGROUND AND LITERATURE SURVEY 3

2.1 Background 3

2.2 Related Works 6

3 SHAP RULE GENERATOR 9

3.1 Calculating SHAP Values 9

3.2 Training Decision Trees on SHAP Values 11
3.3 Node Selection and Filtering 12
3.4 Comparison of the Rules 14
3.4.1 Identification of the Intersections Between Rules 14
3.4.2 Intersection Handling 16
3.5 Creating Rules 18
4 EXPERIMENTAL EVALUATION 21
4.1 Model and Rule Accuracies 21
4.1.1 DDQN 22
4.2 ML Models 24
4.2.1 Iris 25
4.2.1.1 Observation on the Decision Tree 28
4.2.2 Heart Attack 29
4.2.3 Adult Census Income 33
4.3 DDQN Results 35
4.4 Effect of minimum required number of rule for a label 36
5 CONCLUSIONS 39
REFERENCES 41
APPENDICES
A ACCURACY METRICS FOR ML MODELS AND THE EXPLAINER . . 45
A.1 Accuracy metrics for model and SHAP Rule Generator on Iris training dataset 45

A.2 Accuracy metrics for model and SHAP Rule Generator on Heart Attack training dataset 45

A.3 Accuracy metrics for model and SHAP Rule Generator on Adult Census training dataset 45

LIST OF TABLES

TABLES

Table 4.1 ML model results on training set of the Iris 27

Table 4.2 ML model results on test set of the Iris 27

Table 4.3 Feature attributes of the heart attack dataset 31

Table 4.4 ML model results on training set of the Heart Attack 31

Table 4.5 ML model results on test set of the Heart Attack 31

Table 4.6 ML model results on training set of the Census Income 33

Table 4.7 ML model results on test set of the Census Income 34

Table 4.8 ML model results on test set of the "GARAN.IS" stock price history 36

Table 4.9 ML model results on test set of Census Income when minimum required number of rules for a label is 2 37

Table 4.10 ML model results on test set of Census Income when minimum required number of rules for a label is 3 37

Table 4.11 ML model results on test set of Census Income when minimum required number of rules for a label is 4 37

LIST OF FIGURES

FIGURES

Figure 2.1 Overview of SHAP [1]. This depicts the magnitude of affection for each feature on output score on the right. 4

Figure 3.1 Workflow of the SHAP Rule Generator 10

Figure 3.2 Decision Tree trained on SHAP values for Iris Versicolour output class 12

Figure 3.3 Intersection occurs due to FEATURE 1 15

Figure 3.4 There is no intersection between the two nodes/rules 15

Figure 3.5 Four possible scenarios for an intersection to happen for two closed intervals, the intersection interval is highlighted by dotted lines. The intersection is removed from Interval 2 in each case to make them mutually exclusive 17

Figure 3.6 Intersection handling for two rules. Feature 1 is selected to remove the intersection from Rule 2, which has a lower SHAP score. After the intersection is removed (the table below), two rules (i.e. two list of feature intervals) get separated from each other completely 18

Figure 4.1 DDQN implementation overview for stock trading, image is taken from the implementation repository [2] 24

Figure 4.2 Confusion Matrices (rows and columns represent labels, 0=Setosa, 1=Versicolour, 2=Virginica, -1=undecided) for explainers based on four different ML models trained on training set of Iris. Predictions of given models are chosen as ground-truth values. Predictions of the explainers' are used as predicted values in the matrices for evaluation . . 26

Figure 4.3 Black-box model (Decision Tree) on the left, and the key points are detected (exact or very close values) by SHAP Rule Generator on the right (Python code) 28

Figure 4.4 Plot shows the average impact of each feature on labels 29

Figure 4.5 Black-box model (Decision Tree) on the left, boxes mark the similarity between the feature intervals for same labels in the DT and the explainer on the right (Python code) 30

Figure 4.6 Confusion Matrices (rows and columns represent labels, 0=False, 1=True, -1=undecided) for explainers based on four different ML models trained on training set of Heart Attack dataset. Predictions by a given models are chosen as ground-truth values. 32

Figure 4.7 Confusion Matrices (rows and columns represent labels, label 0=False, 1=True, -1=undecided) for explainers based on four different ML models trained on training set of Census dataset. Predictions by a given models are chosen as ground-truth values. 34

Figure A.1 Confusion Matrix for accuracy of explainers based on four different ML models trained on Iris training dataset. 46

Figure A.2 Confusion Matrix for accuracy of four different ML models trained on Iris training dataset. 47

Figure A.3 Confusion Matrix for accuracy of explainers based on four different ML models trained on Heart Attack training dataset. 48

Figure A.4 Confusion Matrix for accuracy of four different ML models trained on Heart Attack training dataset. 49

Figure A.5 Confusion Matrix for accuracy of explainers based on four different ML models trained on Adult Census training dataset. 50

Figure A.6 Confusion Matrix for accuracy of four different ML models trained on Adult Census training dataset. 51

LIST OF ABBREVIATIONS

2D	2 Dimensional
4D	4 Dimensional
ML	Machine Learning
DQN	Deep Q Network[3]
DDQN	Double Deep Q Network[4]
SHAP	Shapley Additive Value Estimator Library[1]
LIME	Local Interpretable models[5]
PDP	Partial Dependence Plot
RuleFit	An explanability method
DT	Decision Tree
SVC	Support Vector Machine
KNN	K-nearest Neighbourhood
LOGREG	Logistic Regression
GARAN.IS	Stock Symbol for Garanti Bank in Yahoo finance
BIST	Borsa Istanbul
LASSO	Least Absolute Shrinkage and Selection Operator

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Explainability is essential for a decision taken by black-box models. There might be a requirement for explaining the decisions taken by those black-box models in industries such as medicine [6], mortgage market [7], and justice [8]. In order to meet those requirements, explanation tools must be present. Otherwise, black-box models cannot be genuinely trustful in such life-critical matters. In order to resolve this problem, scientists have developed various explainability methods. However, most of the explanation tools focus on instance-based explanations. Also, most tools cannot produce an easy-to-read explanation of the models. We present a new approach for extracting decision rules which are easy to read and giving a global description of the model. We set out to learn and show how we can explain black-box models trained for classification tasks computing SHAP (SHapley Additive exPlanations) [1] values for any model on a given dataset.

1.2 Proposed Methods and Models

We have divided our black-box model selection into two main sections for our proposed workflow. Firstly, we train Machine Learning (ML) [9] models that are simple compared to complex neural network-based models. We use the following models as reference logistic regression (LogReg) [10], k-nearest neighbourhood (KNN) [11], Decision Trees [12] and Support Vector Machines [13].

In the latter section, we use a Reinforcement Learning (RL) [14] based model, which

is Double Deep Q Network (DDQN) [4], an improved version of Deep Q network (DQN) [3] in our work. We reason that we have some control over the agent's behavior by choosing the RL-based model. The reason behind the choice of DDQN over DQN is that latter tends to favor overestimated values over underestimated ones [4].

In the end result, we aim to generate an easy to read decision rules in the form of if-then sentences with the help of other explanation tools. We utilize local explanation techniques of the SHAP, and propose a workflow integrating Decision Trees to extract classification rules for interpreting the model globally. We call the overall work as SHAP Rule Generator.

1.3 Contributions and Novelties

Our contributions are the following:

- Extraction of classification rule set on a given ML model for classification task
- Explanation of ML models globally utilising local explainability tools of SHAP
- Generation of a rule set that is ready to execute, trying to explain the black-box model

1.4 The Outline of the Thesis

In the subsequent Chapter 2 we explain our background and literature research related to the topic. Then in Chapter 3, we explain our proposed framework. In the last Chapter 4, evaluation metrics are shared and conclusions are made.

CHAPTER 2

BACKGROUND AND LITERATURE SURVEY

2.1 Background

Most ML models make decisions about given data but do not explain why they have given that prediction. Interpretable ML methods try to resolve this problem. Interpretable ML methods can be defined as "the use of various techniques which may include other machine-learning models for extracting relevant knowledge about black-box structure whose domain relationships are contained in data" [15]. There is an increasing demand for making ML models explainable for various reasons such as legal requirements, human life critical tasks, finance applications, etc. There are ML models which are inherently interpretable thanks to their simple architecture, such as Linear Regression, Logistic Regression, and Decision Trees. However, not every task can be solved with these models. Therefore, advanced algorithms and architectures involve multiple layers of connected network nodes. Although they provide satisfying results for simple tasks, they cannot explain the decisions behind their predictions. Interpretability of the models comes into play when humans need to understand how the model works. There is no such thing as the best explanation model for any given model. The format or methodology of explainability is still open to further development [15]. The most known model-agnostic methods are SHAP [1], Local Surrogate (LIME) [5], Partial Dependence Plot (PDP) [16], Scoped Rules (Anchors) [17].

In our work, we have developed and proposed an explainability method that generates human-readable decision rules for a given ML model. Our proposed approach uses SHAP underneath as a source of truth for finding the most crucial feature intervals for a given target class.

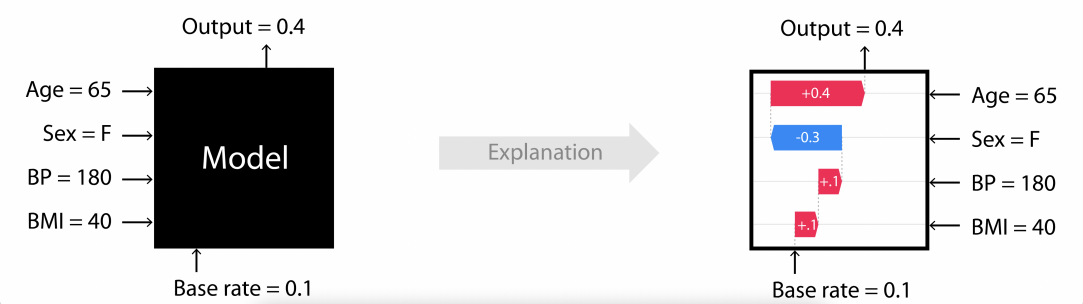


Figure 2.1: Overview of SHAP [1]. This depicts the magnitude of affection for each feature on output score on the right.

SHAP [1], as the authors describe, is a game theoretic approach to explain the output of any machine learning model. SHAP value for a given output class for a given instance is a vector whose indexes correspond to each feature attribute. The values in each index show the magnitude of that feature's effect on the prediction value. In other words, SHAP values tell us the amount a feature attributes to the change in the model's prediction. SHAP accepts an instance and a model then produces an explanation like in Figure 2.1. The higher the value, the higher that feature contributes to the output score with respect to the base value, which is the log odds average of all of the SHAP values for a given output target. The negative SHAP values imply that those features negatively decrease the output score. So features with negative SHAP values are the ones that are against the prediction. In their paper, KernelSHAP, which is model-agnostic, tries to estimate the instance's contributions of each feature value to the prediction proposed. In essence, KernelSHAP uses a black-box model to approximate actual Shapley values, which are time-consuming to compute.

The drawback of this method is the impact on the explanation quality. KernelSHAP computes an approximation of Shapley values efficiently with the assumption that the features are independent. This assumption may not be valid all the time. Aas, Kjersti et al. [18] have proposed an improved version of KernelShap, which consider the case when there is a dependency between the features.

In a method called LIME [5], the researchers have tried to approximate a surrogate model to explain individual predictions. LIME generates a new dataset by the variations around (within the neighbourhood of) the given instance so that the surrogate

model can sense how and why the black-box model made predictions. They compare the predictions of the surrogate model to the black-box model and do some calculations to find prominent feature values. This method has no relationship or interaction with our thesis work. It is just given as a piece of general knowledge.

The partial Dependence Plot (PDP) method explains the relationship between one or two feature values on the predicted outcome. Generally, visual representation between two features is shown. The method is computationally expensive since it considers the whole dataset for a single feature explanation to give a global explanation for the predicted outcome.

$$\hat{f}_S(x_S) = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_S, x_C^{(i)}) \tag{2.1}$$

The partial function $\hat{f}_S$ is estimated by calculating averages in the training data, also called Monte Carlo integration. x_S represents the features we are interested in, the number of items in it is usually one, X_C is all the other features in the dataset, and n represents the number of instances in the dataset.

Q-learning [19] is a reinforcement learning method for finding the best action/value given a state by constructing a table called Q-table. The creation of a Q-table is a computationally heavy job. Therefore estimation methods are used when the dataset is extensive. Q-learning is not directly used in our work due to its computational complexity.

DQN [3] is an implementation for Q-learning [19] by utilizing deep neural networks [20]. Instead of creating a very big matrice for Q-table in the Q-learning, this table is approximated by neural networks. We will not go into the details since our objective is to provide explainability methods. For further information, please address the original paper.

DDQN [4] is an improved version of DQN [3] by utilizing deep neural networks. This method aims to get better results for overestimated values by decomposing the action selection and action evaluation. The target network in DQN is also used for action evaluation. The intuition behind this is using the same network for selection

and evaluation. The target network replaces the Max operation in the original DQN with an extra evaluation step. In double DQN, the online network is responsible for selecting the action, and the target network evaluates the output. The target network is just a periodic copy of the online network.

2.2 Related Works

In this section, we provide our research on the field and would like to share the papers that have influenced our way of thinking.

Akyuz and Birbil propose two methods for rule-based learning, namely rule extraction and rule generation [21]. Rule extraction aims to select optimal rules either from tree-ensemble methods or rule-based methods [22]. They utilize Linear Programming models for these tasks. Their paper mainly focuses on extracting a desired set of rules, and the related work section includes references to the papers around that topic. The similarity between our work is the extraction of the rules from tree-ensemble models. Their work can be used in our SHAP-based DT models to select more appealing rules. This is left as future work.

Shi et al. have utilized the DDQN model for stock prediction in their paper called "Stock trading rule discovery with double deep q-network" [23]. Their work is similar to our DDQN example. Additionally, they practice two reward methods, cash, and accrual, whereas we utilize only one. They compare two of them in various stock indices. The term "rule" in the title of the paper refers to the latent dynamics of the DDQN model rather than the "rule" concept in our work which is "if-then" style decision rule. There is an ambiguity in the word "rule" between the two works.

Attanasio et al. have utilised associative classifiers [24] in stock trading. They have trained association rules based models to generate trading signals [25]. They express in their paper that interpretability is the reason for associative models. Generated rules state the direction according to indicators that are added as a feature to price data. Their rules, however, do not consider the intersection between themselves. We can find one instance that can fall into more than one rule, which can cause ambiguity. We resolve this problem in our work by proposing an intersection handling algorithm.

RuleFit [26] utilizes the linear and tree-based models to give interpretability, in other words, to create rules for a given problem. It uses tree ensemble algorithms, such as RandomForest and XGBoost, to extract many promising rules. Then, the rules are used within the linear model (Lasso) to reduce the size of the rules. The result, in the end, is a sparse model whose features are the original features and the rules.

Our work is similar to RuleFit in some areas. We also use tree ensemble methods to generate trees. Although the intuition behind using trees might be similar to ours, we generate the output class size of many trees as opposed to RuleFit may generate an arbitrary number of trees. The idea of generating rules from decision trees is similar. Our tree's target values are SHAP, whereas RuleFit tries to predict actual values.

After finding out the rules are entirely different between our work and RuleFit, we reduce the number of rules in each tree separately by applying some manual work. In contrast, RuleFit uses Lasso [27] for this purpose. We compare each rule to find out intersections aiming to predict better results. The RuleFit can handle classification and regression tasks. However, our model only supports classification problems. Also, RuleFit is an inherently explainable model and does not aim to explain any black-box structure as our work does.

Scoped Rules (Anchors) [17] uses reinforcement-based methods to search for simple if-then rules similar to our work. Like LIME, it also tries to generate local explanations. There are two methods explained in their papers to find out rules, and the first one is called bottom-up, which starts with if-then rule with individual features and adds more feature intervals one by one until it reaches a good precision. As the features are added to the rule, the coverage (i.e. the number of instances that belongs to/follow the rule) drops. In the end, lots of anchors (rules) are given to the user.

Our proposed method, SHAP Rule Generator, shares the similarity of finding out if-then rules like in the RuleFit. RuleFit does not explain the model globally. Instead, they give intuition to the user on how the model behaves for various kinds of instances. On the other hand, our work aims to explain how the model behaves globally.

CHAPTER 3

SHAP RULE GENERATOR

In this chapter, we explain the SHAP Rule Generator to extract rules for a given classification model. We break down our pipeline into six main sections. In the first step, we choose an ML model we want to explain and train on a dataset. Secondly, we calculate SHAP values for the trained ML model along with the data. We extract the most critical feature intervals using SHAP values with the help of Decision Trees [12]. Later, we extract the most prominent feature intervals. Next, we compare them against intersection possibility and build a set of rules to determine an instance's class. In the last step, we combine the feature intervals we have found to build if-then rules.

The whole pipeline of the SHAP Rule Generator can be summarised in Figure 3.1.

This chapter explains the main blueprints of the models and algorithms. Evaluation metrics and statistical data are provided in Chapter 4.

3.1 Calculating SHAP Values

We have chosen various ML models for our black-box model, such as Logistic Regression [10], K-Nearest Neighbourhood [11], Decision Trees [12] and Support Vector Machines [13]. We will share the evaluation metrics for the models in the subsequent chapter.

After completing the training, we use KernelSHAP to get SHAP values for each feature value of all instances for each output target. KernelSHAP expects a model and a dataset as input to produce Shapley Values. Assuming we have a feature size of R and instance size of X and Z many output classes, our resulting SHAP values

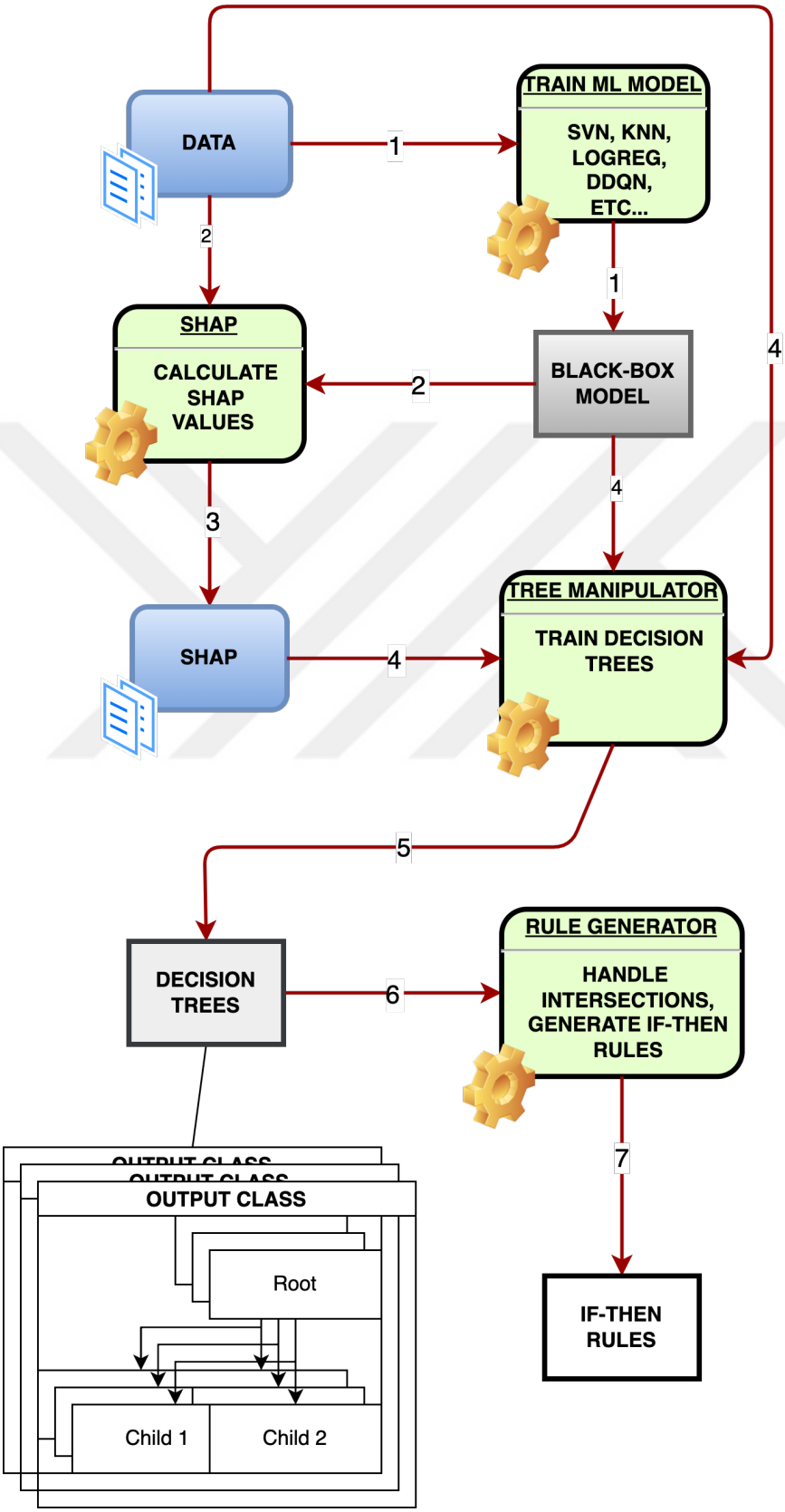


Figure 3.1: Workflow of the SHAP Rule Generator

vector would have $X \times R \times Z$ many data points. These SHAP values will be used to determine the critical feature levels for our rule sets. We have not used any Hyperparameter tuning techniques for the black-box models. We are not interested in discovering the models that can make good decisions. We are only interested in their explainability of them.

3.2 Training Decision Trees on SHAP Values

We want to determine the feature areas which contribute the most to the selected output class. We select DT as a tool to determine key intervals for features with higher SHAP values. By an interval, we mean a closed line segment representing a set of real numbers within a given boundary. Our target is to extract the most promising intervals for each feature per label.

We train unique DTs for each output target/label with corresponding SHAP values for a given instance. Hence, each DT tries to approximate SHAP values for an instance set in which all instances have the same label. We specifically want to overfit (i.e. the model memorizes the dataset) the Decision Trees because we can get the key levels in given data only. In other words, we would like to find out the if-then relationship in the existing data only, generalization of the DT in our case does not make sense.

Figure 3.2 shows a Decision Tree trained on SHAP values for the Iris Versicolour output class. Leaf nodes' values in the tree represent predicted SHAP values by the decision tree according to instances whose feature values belong to the rules following the path up until the root. In other words, leaf nodes represent that instances that belong to the decision rules above have SHAP values given in the value section of the node. We understand that the higher SHAP values contribute the most to the given output class; therefore, if we select those nodes with higher SHAP values, their decision rules will contribute the most to the target score. We train decision trees for each output target class, and in the subsequent section, we filter out some nodes and extract decision rules for them. There is no data preprocessing for the SHAP values since they are already in log odds units.

Instead of training Decision Trees for once, we use a Hyperparameter tuning tech-

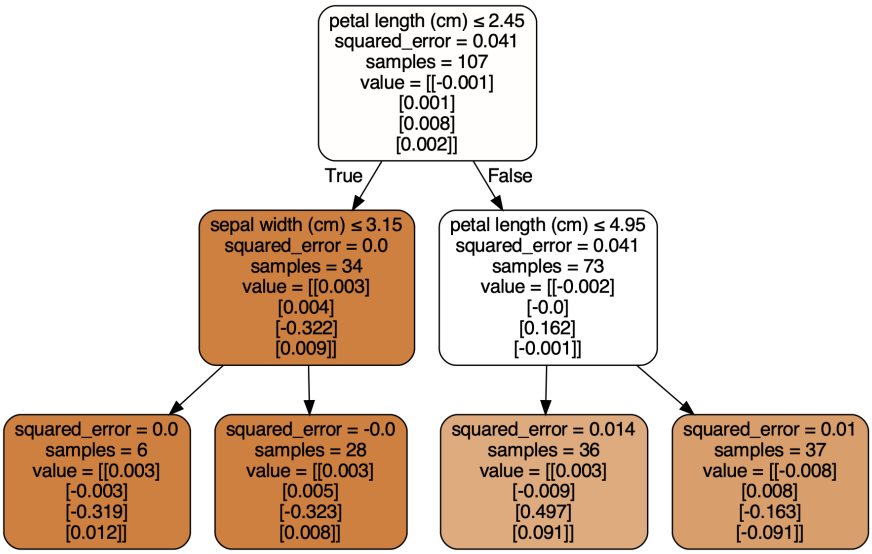


Figure 3.2: Decision Tree trained on SHAP values for Iris Versicolour output class

nique called Grid search [28]. In this method, we provide a list of parameters that the Decision Tree can accept and Grid search trains unique Decision Trees for every possible combination of all the parameters. We have tuned the "max_depth" parameter of DT with the following range 1..3. We limit the depth of a tree to three all the time. Nevertheless, this number can be changed as a hyper-parameter depending on feature size. If we set this number as constant, we might extract rules that contain so many feature intervals and there might be lots of rules due to the massive amount of leaf nodes that are generated. On the other hand, this limit of the maximum depth of a tree poses a threat to our explainer's correctness. However, we need to make this sacrifice in favor of interpretability. A model with high accuracy is taken to be used in our rule generation process.

3.3 Node Selection and Filtering

We do data preprocessing in this step. We find the maximum and minimum values for each feature in the dataset in order to set limits in our if-then rules. Otherwise, the unforeseen overestimates might degrade our explanation performance. We must

assign a list of feature intervals to each leaf node to represent the node's decision rule. We deduct a simple if-then rule for each leaf node in each decision tree by taking a dot product of feature interval in each parent node up until the root. By taking the dot product, we mean concatenation of feature intervals by an and operator in a logical sense. Feature interval in any node means its parent's decision when we want to follow a path from the parent to the node.

We can look at an example from our work to better visualize the concept. In Figure 3.2, we assign the following statement to the bottom left leaf node as $[[" sepal_width(cm)", (2, 3.15)], [" petal_length(cm)", (1, 2.45)]]$, value 1 is the minimum value for petal length and 2 for the sepal width feature in the dataset. The corresponding if-then rule would be the concatenation of the two feature intervals as the following:
$(2 <= sepal_width(cm) <= 3.15)\ and\ (1 <= petal_length(cm) <= 2.45)$

In the last step, we have a bunch of feature intervals for each leaf node in our tree. In DTs, the same feature might be repeated twice in a path from the root to a leaf. This can cause many feature intervals with the same feature but different intervals. In order to reduce the number of feature intervals assigned to a node, we combine and organize the list of intervals whenever we have similar intervals for the same feature in a node. There might be overlapping feature intervals for the same feature. We merge them to have fewer items in our leaves. Therefore, we merge the feature intervals if one covers another within a node. We combine them if they intersect at a point or at an interval.

The leaves in the decision tree in Figure 3.2 represent a single output class, Versicolor. However, we are not sure for each node whether the model predicts any instance as Versicolour meets the requirements given in the node. In this example, we eliminate some leaf nodes if there is no instance from the dataset whose predicted class is the same as the decision tree, Versicolour. The intuition behind this is that we want to find feature intervals that represent the output class with high coverage, the number of instances that the rule represents within the dataset according to the output class predicted by the original black-box model. If no such instance belongs to the if-then rule, it does not make sense to include it since the rule does not have high coverage

and does not mimic what the model does correctly.

The result for a single node represents an if-then rule with one or multiple feature intervals concatenated by an and operator. At this point, the single if-then rule states the following, if an instance's feature values fall within the feature intervals described by the rule, that instance is labeled as the decision tree's assigned label, which is a label in the original dataset. Hence, we have a bunch of rules stating that if the rule covers the instance's feature values, they belong to that label.

We gather a list of if-then rules from each tree by using the leaf nodes only. Each tree represents a category in our classification problem. Therefore each list of rules belonging to the same tree represents/assigns the same category to the tree they belong to.

3.4 Comparison of the Rules

At this point, we have a bunch of if-rules. Each assigns a label if a given instance complies with the feature interval values defined in that rule. We know those feature intervals in leaf nodes of the same decision tree do not intersect with each other. The same rule also applies to our trees, so this infers that the leaf nodes (we may call just rules) in the trees do not intersect, provided they belong to the same tree. However, they may intersect with the rules in other trees since there is no such guarantee. We need to find out the nodes (rules) that are intersecting, and we should gracefully resolve the conflicts. We propose a unique solution to this problem. However, as we will see in the following sections, our rules might not be separated from each other entirely. There might even be overlapping feature interval areas between the rules, and it might introduce ambiguity in the label of an instance.

3.4.1 Identification of the Intersections Between Rules

As we have pointed out previously, two rules from different trees might have intersecting feature intervals. In other words, given two rules can cover the same areas where an instance falls into and creates confusion around the assignment of its label.

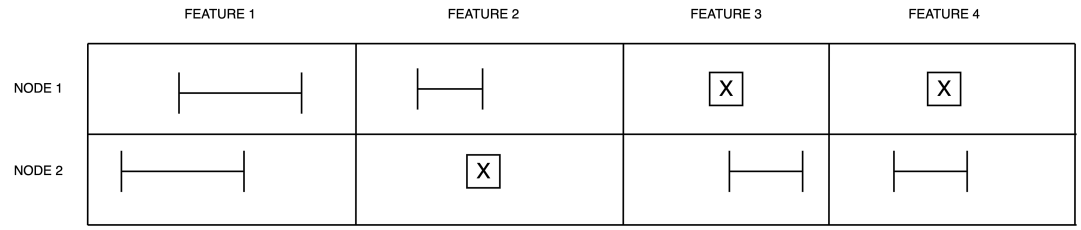


Figure 3.3: Intersection occurs due to FEATURE 1

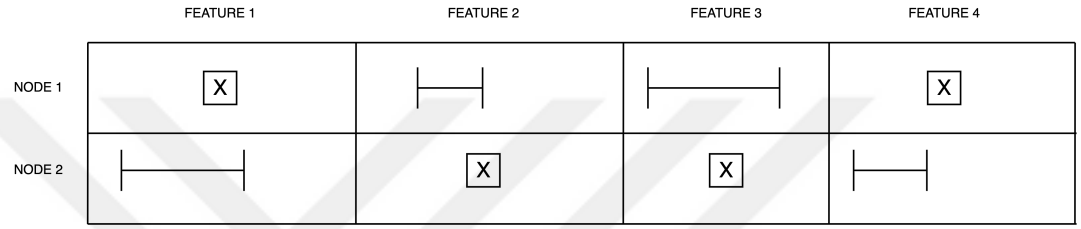


Figure 3.4: There is no intersection between the two nodes/rules

The condition for an intersection between two rules (list of feature intervals) is simply defined as the following, if we can find at least one instance that might fall into more than one rule provided that there is at least one intersecting feature interval in both of these two rules then we say these rules intersect with each other. We can give the following example in Figure 3.3. Feature 1 intervals in both of these rules cause an intersection. If the intersection is resolved somehow, then we would make these two rules mutually exclusive, meaning that there would be no such instance that falls into both of these rules. This is a trivial task to resolve for the example given in Figure 3.3.

The case, which is non-trivial to solve, in Figure 3.4 depicts that an intersection can occur where two rules have completely different feature intervals. We do not have sufficient information to resolve the conflict in that case. Many cases can occur between the two rules in Figure 3.4. They can diverge or overlap with each other.

We can make an analogy to explain the non-trivial case. The intersection problem we are trying to solve is similar to finding an intersection of two planes in 2D with unique dimensions in a 4D world. Planes represent our rules and dimensions as our features. Instances exist in 4D world, meaning they can have up to four feature attributes at most. There might be instances from a dataset that can be in two planes, but we are not

sure enough to tell if these two planes intersect due to insufficient information. Those two planes can define themselves with respect to their dimension attributes and do not tell anything about other dimensions. However, our data can contain more attributes. Since the dimension of rules is less than an instance's dimension, we cannot tell if those rules intersect or not. We categorize those rules as mutually exclusive, which is the trade-off we need to make to generalize our explainer. Otherwise, we would end up with so many rules that we also need to consider the situations where these two planes (rules) intersect in a point, in a line or being coincident and this would populate the number of rules that we have. Another idea introduced by us but not implemented is that by looking at the given data, the position of the planes (i.e. feature intervals of the rules) can be minimized by adding extra feature attributes to make them diverge. Therefore, we would be able to make our rules mutually exclusive within the dataset. We leave this as future work.

We can give a simple algorithm for deciding an intersection for the trivial case mentioned above. If there is at least one same feature in both rules and if they do not intersect or features in the rules are entirely different, then there is no possibility of an intersection. Otherwise, we need to develop a strategy to resolve the conflict. The method for resolving the intersection is explained in the next chapter.

3.4.2 Intersection Handling

Because of an intersection, our explainer may assign an instance to more than one output label due to two similar rules from different trees existing in our explainer. Moreover, this can be confusing because it fails to explain some instances. In other words, some feature interval areas are not correctly explained.

We compare each rule in one decision tree to all the other rules in other trees. Since we know nodes/rules in the same tree are mutually exclusive, we do not need to search for an intersection in the same tree. While comparing the rules against the ones in different trees, we check if there is an intersection like we explained in previous sections. If that happens, we need to remove intersected part from one of the rules to resolve the conflict. Then the question is which rule we should select to extract the intersection. We answer this by the following work.

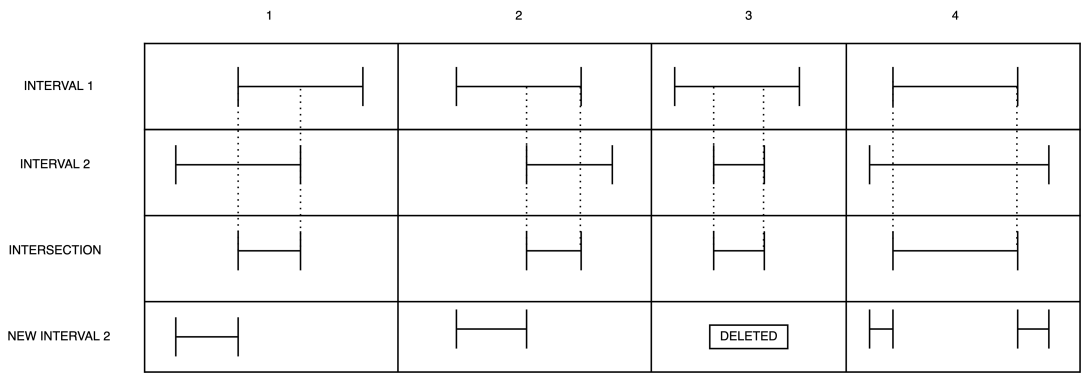


Figure 3.5: Four possible scenarios for an intersection to happen for two closed intervals, the intersection interval is highlighted by dotted lines. The intersection is removed from Interval 2 in each case to make them mutually exclusive

We propose a concept of a "score" assigned to rules which is a sum of SHAP values for a given instance. We sum the SHAP values in each rule to assign a score to a rule. We know from a decision tree that each node has an estimated SHAP value since we trained. We sum each feature SHAP value in a given node to generate a score for a node. Therefore, the higher the score, the higher the magnitude of affection for a feature in the prediction process.

Four possible situations can happen between two intervals when an intersection occurs. We visualize this in Figure 3.5. We compare each feature interval against the other between the two rules. Furthermore, if there is an intersection, as described above, we select one feature interval intersecting. There can be more feature intervals that can intersect but resolving only one of them is enough to make them diverge. As it can be seen from Figure 3.5, we keep Interval 1 always the same in each of the four cases. We find the intersection between Interval 1 and Interval 2 and remove the intersection from Interval 2, assuming that Interval 2 belongs to a rule with a lower "score".

We can also see what happens after the intersection is handled between the two rules in Figure 3.6.

One thing to emphasize is that there is a possibility that after the deletion of rules, some labels might not have any rule, so our explainer cannot predict any instance of those labels. This can threaten our explainer

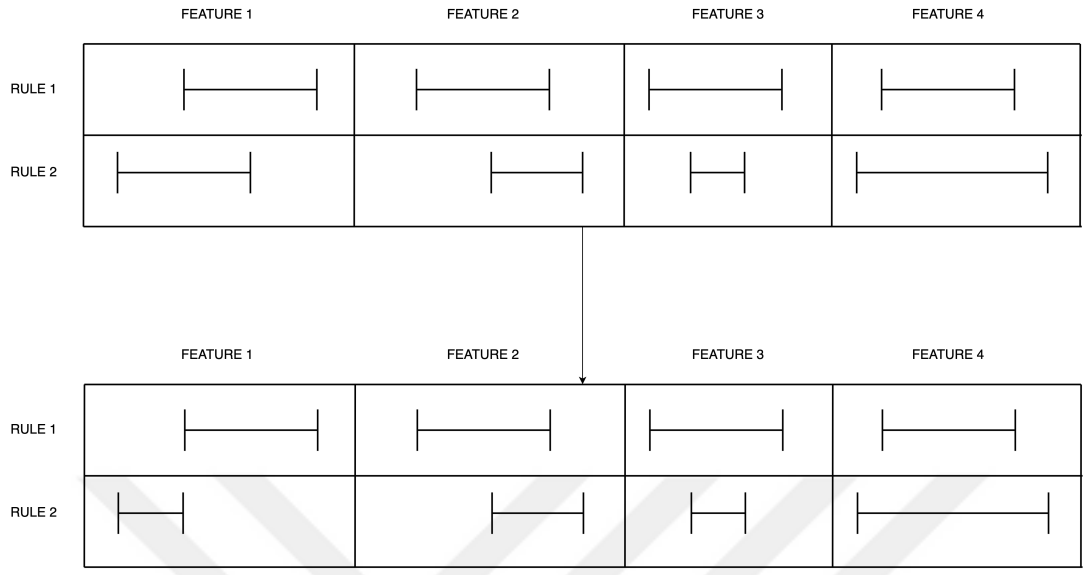


Figure 3.6: Intersection handling for two rules. Feature 1 is selected to remove the intersection from Rule 2, which has a lower SHAP score. After the intersection is removed (the table below), two rules (i.e. two list of feature intervals) get separated from each other completely

since it lacks the ability to categorize an instance into a label. Therefore, we limit the minimum number of rules for each label as 1. Even if a rule is chosen to be eliminated after comparison with another rule from different label, we check if there is any other rule for that label. If there is no such rule, then we keep this rule as is. Even though we know there is a chance that some instances might fall into more than one label, we leave that one rule as is. In future work, a better algorithm can be designed for this specific case.

3.5 Creating Rules

We follow an object-oriented approach for the parts we have created in Figure 3.1. As an example, there are two boxes named "RULE GENERATOR" and "TREE MANIPULATOR" in Figure 3.1. For those, we have created two classes called "ShapRuleGenerator" and "TreeManipulator" in our code repository, which accept the parameters depicted in Figure 3.1. We have also created other helper objects such as "Node" and "Interval" to represent data points in an object-oriented approach. The code

repository of SHAP Rule Generator can be reached via the link shared in the last chapter.

We have all the rules that we need. In this last step, we create an executable Python [29] code as a string which can be passed into exec() [29] [1] command, which executes the given text as a program, to evaluate our explainer. The reason we choose python is because of its readability. Simplicity in the language allows people to read rules effortlessly.

We take each rule one by one and concatenate string snippets embedding the feature intervals in the rules. Each if-then rule is first grouped according to its labels, and for each label, we have an if-then block, each rule within the same label is concatenated by an ‘or‘ clause. We show an example if-then rule set in the next chapter.

[1] https://docs.python.org/3/library/functions.html#exec

CHAPTER 4

EXPERIMENTAL EVALUATION

We have followed an object-oriented approach to utilize our test environment. We have created a class called Shap Rule Evaluator which accepts parameters such as ML model, dataset, and Hyperparameters for the purpose of the test running. The tool we created automatically produced results in tabular format and visual representations. This helped us save lots of time throughout the evaluation process.

4.1 Model and Rule Accuracies

We train an ML model for a classification task on a dataset. Before training, normalization, such as scaling values into the [0,1] range, is applied as a data processing technique for better results depending on the dataset. We have used a split ratio of 15% for dividing the dataset into training and test (validation) clusters in all datasets except the stock prices. This split ratio was unnecessary in our case since we wanted to explain the model's behavior. Our explainer does not care if the model performs poorly or overfits. Its task is fulfilled as long as the explainer successfully predicts the original model. However, we divide the data into training and test sets to follow the convention in the literature.

In this section, we share some metrics for our model and the rules generated by our workflow. We provide accuracy for each model and our explainer for training and validation datasets. We have used a 15% split ratio with shuffling to divide a given dataset into training and validation sections. Since its dataset is time-series, the shuffling method is not utilized in our DDQN example. Unless the other is stated, all class labels/targets are numbered starting from zero and assigned by the order given

to each dataset explanation below. If we have two labels for a given dataset, we name them classes 0 and 1. Also, we have used a minimum number of required rules as one throughout the evaluation if the other is not stated. We will share the results for the DDQN model and classification-based black-box models in two different sections.

We calculate the similarity between a model and the explainer as a key metric to identify our performance on explainability. Similarity means the percentage of the same predictions between two models (the black-box model and the explainer in our case) to all predictions. In short, it aims to calculate how well the explainer replicates the black-box model within each class.

We visualize confusion matrices [30] in order to give more detailed results in this chapter. There might be cases where an instance does not fall into any given rule set. In other words, our rule set does not include any such rule that the instance falls. For this reason, we include an extra label -1 in all confusion matrices to represent undecided predictions made by SHAP Rule Generator.

Frankly, we are more interested in the similarity of our explainer against the actual model rather than the accuracy of a model nor the accuracy of our explainer. We are more interested in the degree of closeness between the actions of the explainer and the black-box model. Therefore, we provide class-level similarity visualizations in this chapter. Nevertheless, the accuracy of the model matters. Otherwise, explaining a model that has not been trained sufficiently would not be very helpful. We provide the model and explainer accuracies based on ground truth values in the Appendix. In this chapter, we use the word "explainer" to refer to SHAP Rule Generator. Therefore we can use the words interchangeably.

4.1.1 DDQN

Algorithmic trading is a way to achieve gains for determined periods in financial markets via computers called robots in the jargon. These algorithms can be constituted by a variety of tools and trading strategies. The robots can fetch data from different markets to play for arbitrage profits or analyze the order book to take immediate action to gain more or stop losses, or they can utilize technical analysis tools to achieve

the maximum profit. The latter approach's problem is that so much noise or features are being used when training the model, and it is not easy to extract information on how the model works.

The problem with the ML models in the finance sector is that the robots (models) become completely black-box models. They do not explain why they took their positions at a given time. In this work, the reasons behind the taken positions by the robots are investigated by first creating a deep learning model and their behaviours are interpreted to generate a code piece that mimics the black-box model.

We have used DDQN [4], which is inspired by the reinforcement learning techniques on a stock price dataset. The reason why we have selected this is that the model allows for a custom reward method to choose which is utilized. Another reason is that after our explainer produces information about the model, the user can modify the network to train in the way a user wants. Our proposed explainer works for a classification task, so we modify our dataset and also our code.

We have used this implementation as a framework for our work [31]. Our dataset is stock prices fetched from yahoo finance APIs by a tool yfinance [32]. We have used TA-lib [33] for enriching our data with technical analysis indicators. In addition to the original DDQN implementation, we have embedded a tool called Backtesting [34] for the calculation of reward and statistical analysis of the trades being made. It also provides a visualization of the trades made on a graph.

Our explainer works on classification problems. Therefore, we interpret the result of the model's output for a given instance in terms of discrete values by selecting the maximum item in the result vector. We define the labels as the following, 0=HOLD, 1=BUY, 2=SELL. The agent can be in only one of the three states while the training continues. According to decisions that agents make, the model calculates a daily reward by calculating a profit (positive) or loss (negative) in the current position. The overall structure can be seen in Figure 4.1

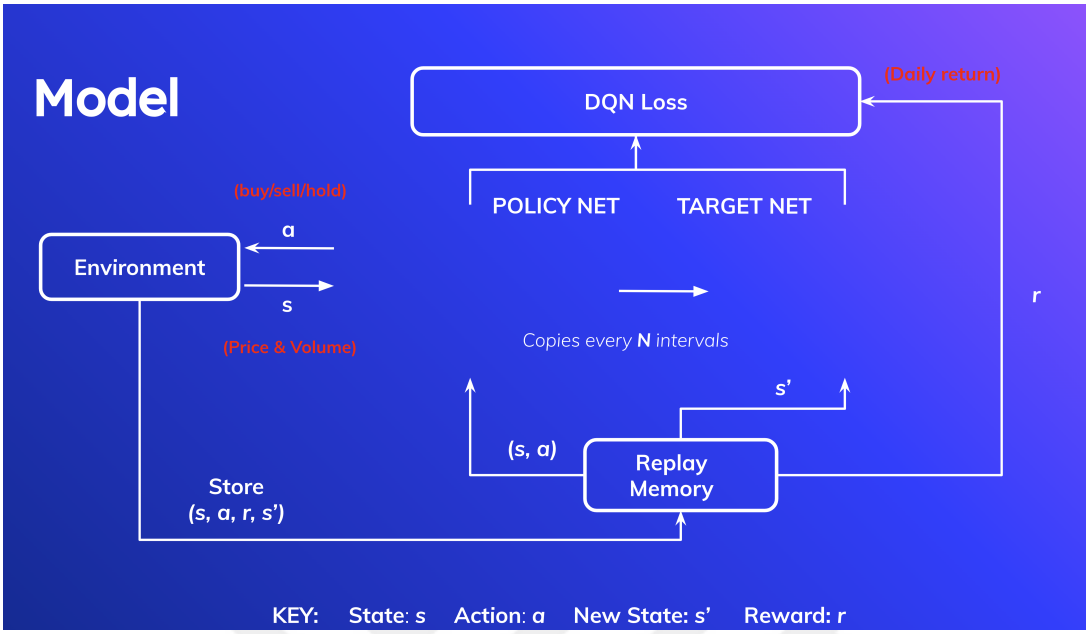


Figure 4.1: DDQN implementation overview for stock trading, image is taken from the implementation repository [2]

4.2 ML Models

Decision Tree, K-nearest Neighbors (KNN), Logistic Regression classifier (LogReg), and Support Vector Machines (SVC) are trained on the following datasets. For our ML models, scikit-learn [35] package is used. We have selected the Iris, Heart Attack, and Census Income datasets. We have split our datasets by the ratio of 15% as training having 85% and test as %15. Our ML models are trained on the training set and validated against the test dataset. We provide overall accuracies for our models and explainer along with the accuracies for each class on its own, running against the ground truth values in a given dataset.

We follow the same convention in each table in the subsequent sections below. The following details explain the column attributes for the tables we have generated. model_name describes the name of the black-box ML model trained as a reference.

We calculate the model_acc referring to the accuracy, in terms of percentage, of the model on a given dataset against the ground-truth values of a given dataset.

$$\text{model_acc} = \frac{\text{correct classifications made by an ML model}}{\text{all classifications}} \times 100 \tag{4.1}$$

We calculate the rule_acc as the accuracy of the explainer generated by SHAP Rule Generator on a given dataset against the ground-truth values of a given dataset.

$$\text{rule_acc} = \frac{\text{correct classifications made by extracted rules}}{\text{all classifications}} \times 100 \tag{4.2}$$

We calculate the similarity as percentage of same class predictions between the explainer and the based model in a given dataset.

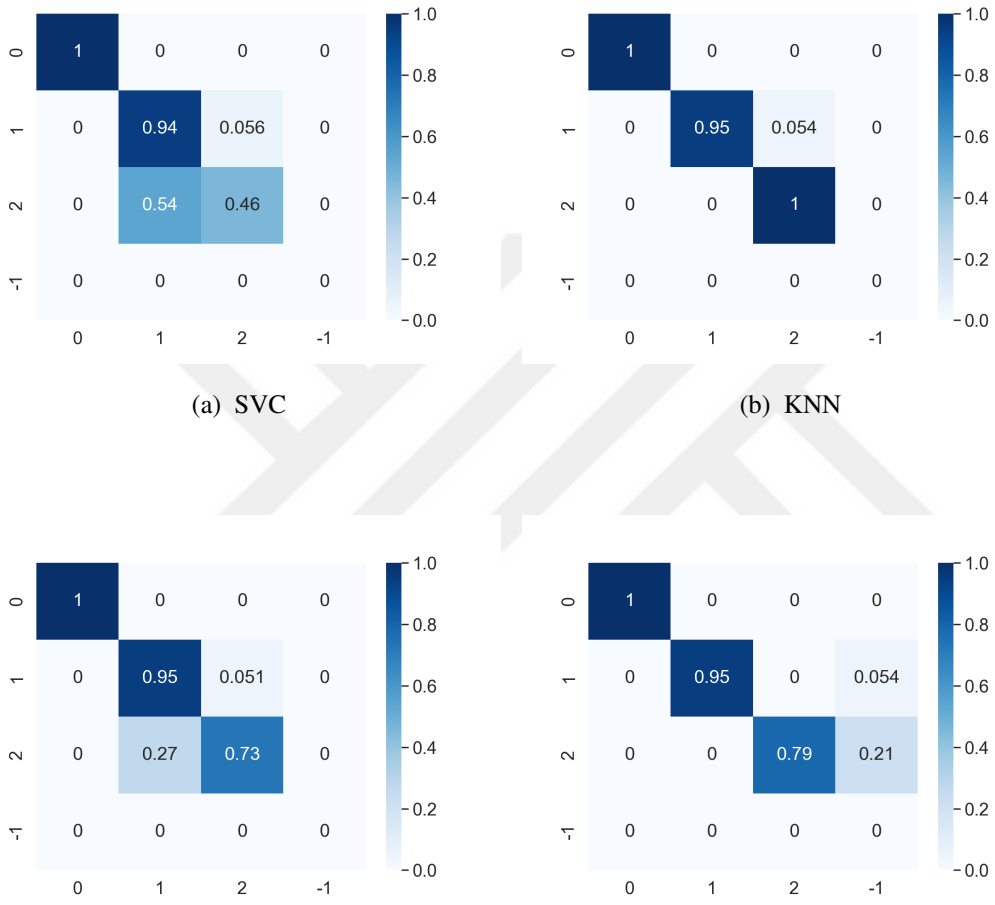
$$\text{similarity} = \frac{\text{the same classifications for explainer and an ML model}}{\text{all classifications}} \times 100 \tag{4.3}$$

Accuracy metrics we share in the equations 4.1, 4.2 and 4.3 might give misleading results. One of the reasons for not choosing accuracy might be due to unbalanced classes. However, this is not the case in each dataset we have used. The main problem with the accuracy is that the full picture is not represented. Model or our explainer might find the correct labels for one class out of two all the time and fail at the other, provided the data is balanced. In this case, accuracy is 50%, but this does not show that our explainer gives a promising result on the first label. Therefore, we also provide accuracy per label in terms of confusion matrices.

4.2.1 Iris

Iris dataset includes a total of 150 instances. Dataset is publicly available, also SHAP library provides it inside. We have the following attributes sepal length in cm, sepal width in cm, petal length in cm, petal width in cm and we have three classes, Iris Setosa (Class 0), Iris Versicolour (Class 1), Iris Virginica (Class 2).

We share accuracy metrics for each explainer (SHAP Rule Generator) running on each ML model in Tables 4.1 an 4.2. Also, in Figure 4.2 confusion matrix for SHAP Rule Generator explains the ML models trained on the training set of Iris.



(a) SVC

(b) KNN

(c) DT

(d) LogReg

Figure 4.2: Confusion Matrices (rows and columns represent labels, 0=Setosa, 1=Versicolour, 2=Virginica, -1=undecided) for explainers based on four different ML models trained on training set of Iris. Predictions of given models are chosen as ground-truth values. Predictions of the explainers' are used as predicted values in the matrices for evaluation

Table 4.1: ML model results on training set of the Iris

model_name	model_acc	rule_acc	similarity
DT	100.0	88.9	88.9
KNN	95.3	95.3	98.4
SVC	97.6	75.6	77.9
LogReg	96.8	89.7	90.5

Table 4.2: ML model results on test set of the Iris

model_name	model_acc	rule_acc	similarity
DT	100.0	100.0	100.0
KNN	86.9	78.2	91.3
SVC	100.0	100.0	100.0
LogReg	100.0	95.6	95.6

It can be observed that the similarity column in Table 4.1, represents the average values of the diagonal axis in Figure 4.2. This condition holds true for all of the confusion matrices we calculate in this chapter.

Looking at the similarity column in Table 4.1 and 4.2 we can say that our explainers have given good results both on training and test sets. The SVC model in Table 4.2 has performed worse than others with respect to the similarity column. The reason is that extracted rules have failed to choose correct labels for instances belonging to class 2, 54% of the instances in class 2 are left undecided, and hence the accuracy has dropped for that specific explainer. The explainers tend to have 10% less accuracy on average compared to the model. We consider this result promising since there is an information loss in the process of extracting rules. Considering that we are trying to explain complex models with only if-then rules, we conclude that our explainers have performed well for Iris.



Figure 4.3: Black-box model (Decision Tree) on the left, and the key points are detected (exact or very close values) by SHAP Rule Generator on the right (Python code)

4.2.1.1 Observation on the Decision Tree

We also would like to point out an observation we have made for "Decision Tree" model in Table 4.1. Since DT model is easy to interpret we can extract rules and compare against our explainer.

It can be seen from Figure 4.3 that the Decision Tree and the explainer for label 0 (Iris Setosa) have found their own way of predicting. The DT uses petal_width feature. On the other hand, the explainer utilizes petal_length for the decision. We investigated the dataset and concluded that both decision rules seemed correct. The reason for the different choice of feature is due to SHAP. SHAP found out that petal_length has affected the output result more than the other features for the class label 0. We can

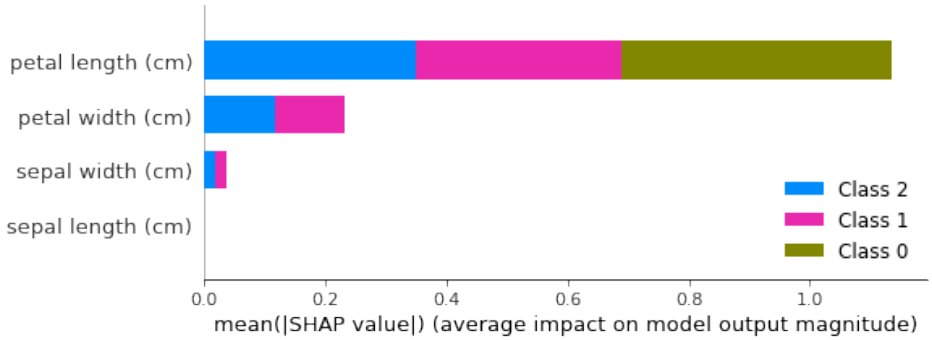


Figure 4.4: Plot shows the average impact of each feature on labels

assert this statement from the summary plot in Figure 4.4 for the SHAP values. The feature "petal length" has more impact for class 0 (Iris Setosa) than other features for the model.

This result shows and we also confirm that our explainer is very limited by the trustfulness of the SHAP. If the KernelSHAP cannot produce good results, it can degrade our explainer's performance. Unfortunately, there are no key metrics that we can use to test/analyze KernelSHAP. This is yet another undiscovered point in the literature. KernelSHAP values are meaningful as long as the person using them understands and asserts the results and extracts relevant information. Since it is an approximation tool for Shapley Values, there is a possibility of discrepancies in the numbers compared to original Shapley Values, which are compute-intensive.

We can show that our explainer not only found the critical feature intervals but also predicted the labels correctly, just like the DT. This behaviour can be seen in Figure 4.5

4.2.2 Heart Attack

The heart attack dataset contains information about the patients with higher heart attack tendencies [36]. The given task is a binary classification problem, 0 = less chance of heart attack 1 = more chance of heart attack. The dataset contains 303 data instances. Before training, we normalized the feature attributes within the range of [0, 1]. Feature attributes for the dataset are given in Table 4.3.



Figure 4.5: Black-box model (Decision Tree) on the left, boxes mark the similarity between the feature intervals for same labels in the DT and the explainer on the right (Python code)

Table 4.3: Feature attributes of the heart attack dataset

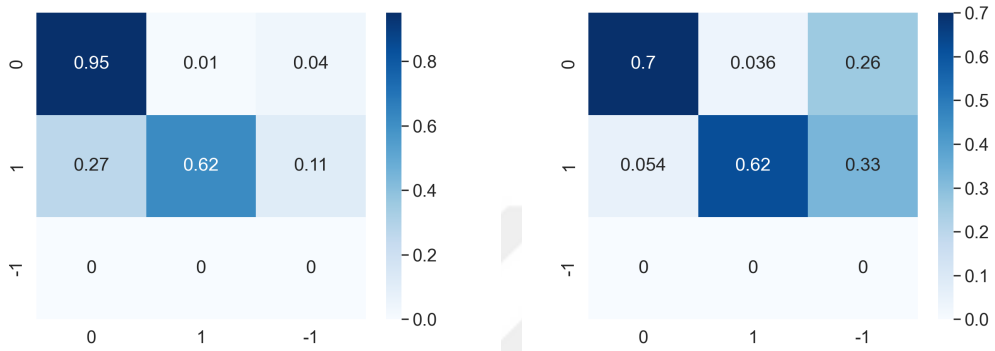
Age	Age of the patient
Sex	Sex of the patient
exang	exercise induced angina (1 = yes; 0 = no)
ca	number of major vessels (0-3)
cp	Chest Pain type chest pain type (1=typical angina, 2=atypical angina, 3=non-anginal pain 4=asymptomatic)
trtbps	resting blood pressure (in mm Hg)
chol	cholestoral in mg/dl fetched via BMI sensor
fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
rest_ecg	resting electrocardiographic results (1= normal, 2=having ST-T wave abnormality, 3= showing probable or definite left ventricular hypertrophy by Estes' criteria)
thalach	maximum heart rate achieved
target	0= less chance of heart attack 1= more chance of heart attack

Table 4.4: ML model results on training set of the Heart Attack

model_name	model_acc	rule_acc	similarity
DT	100.0	70.8	70.8
KNN	88.7	62.2	65.3
SVC	84.8	70.8	74.7
LogReg	84.4	75.1	82.8

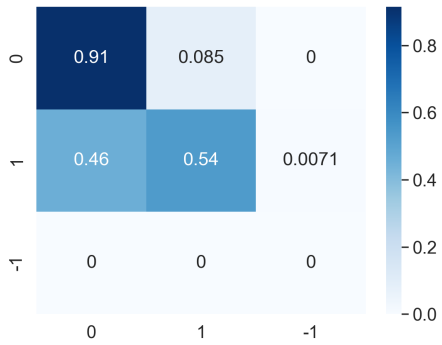
Table 4.5: ML model results on test set of the Heart Attack

model_name	model_acc	rule_acc	similarity
DT	100.0	82.6	82.6
KNN	89.1	45.6	43.4
SVC	86.9	82.6	86.9
LogReg	89.1	67.3	71.7

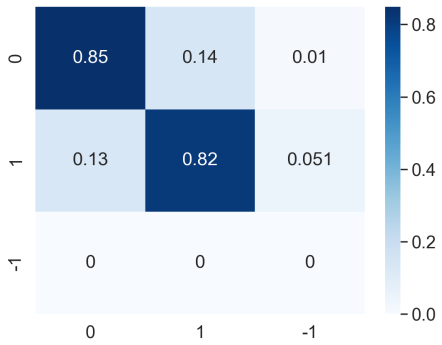


(a) SVC

(b) KNN



(c) DT



(d) LogReg

Figure 4.6: Confusion Matrices (rows and columns represent labels, 0=False, 1=True, -1=undecided) for explainers based on four different ML models trained on training set of Heart Attack dataset. Predictions by a given models are chosen as ground-truth values.

Table 4.6: ML model results on training set of the Census Income

model_name	model_acc	rule_acc	similarity
DT	97.9	76.0	77.5
KNN	87.4	80.8	85.9
SVC	84.2	53.4	56.2
LogReg	82.6	41.5	43.6

Looking at Table 4.4 and 4.5, we can say that our explainer has made promising results both on training and test sets. Also, the diagonal values in Figure 4.6 and similarity column in Table 4.4 show that the models except the KNN have performed well per label. It is clear that model accuracies have dropped due to complexity in the dataset with respect to the Iris dataset. On the other hand, our explainer has maintained good results despite the feature and instance size increase in the dataset compared to Iris.

4.2.3 Adult Census Income

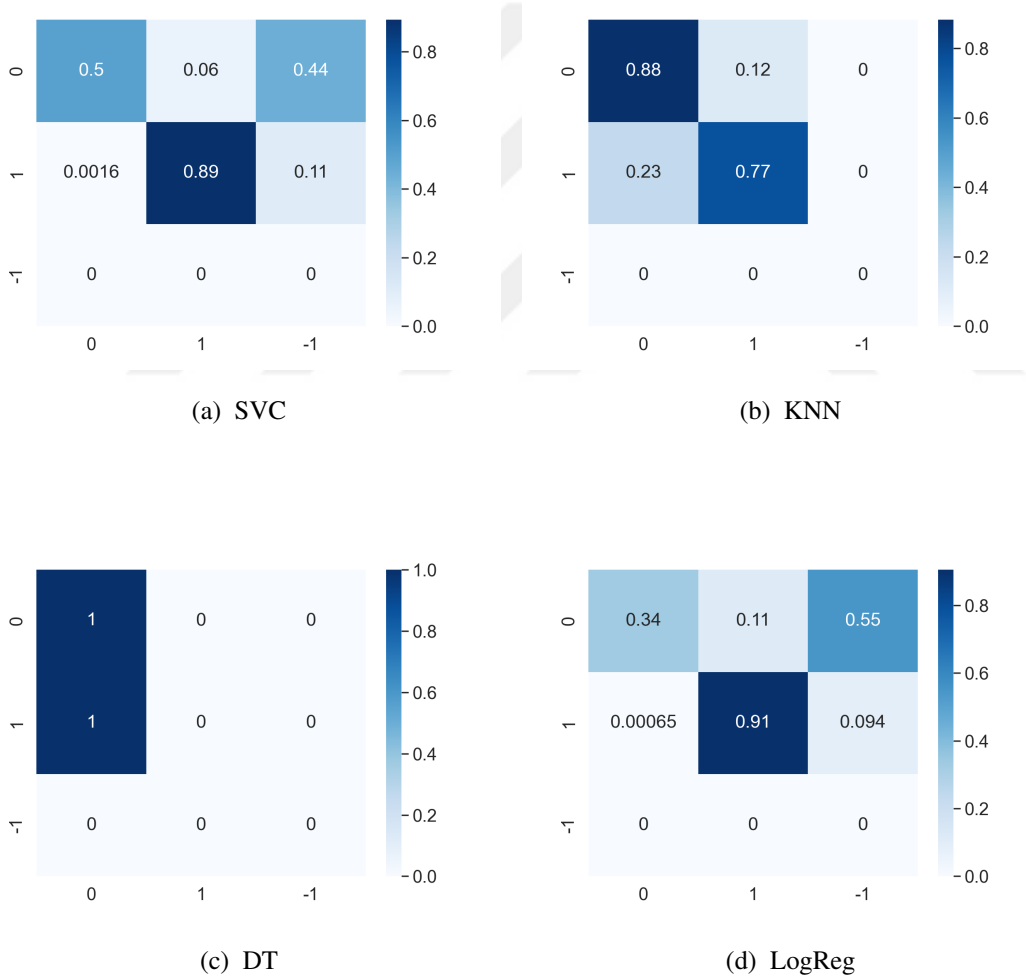
The dataset was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics) [37]. The classification task is to determine whether a person makes over $50K a year [37]. In our training set includes the following feature attributes: 'Age', 'Education-Num', 'Sex', 'Capital Loss', 'Occupation', 'Race', 'Capital Gain', 'Country', 'Hours per week', 'Marital Status', 'Relationship', 'Workclass'.

We have a total of 32561 instances in the dataset. Training set includes 26048 and the test set includes 6513 instances. We have normalized the following attributes Age, Race, Capital Gain, Capital Loss and Hours per week to a range of [0, 1].

The accuracy and similarity results in Table 4.6 and 4.7 have plunged compared to previous datasets. Possible reasons for this might be the increase in dataset and feature sizes. Another reason we found out was that when we took a closer look at the rule class similarity column, the class 0 score tended to be lower than class 1. We have investigated this issue and found that our minimum required number rule constraint

Table 4.7: ML model results on test set of the Census Income

model_name	model_acc	rule_acc	similarity
DT	99.0	64.3	63.5
KNN	86.3	36.5	33.6
SVC	83.3	61.9	56.7
LogReg	82.2	59.5	56.7



(a) SVC

(b) KNN

(c) DT

(d) LogReg

Figure 4.7: Confusion Matrices (rows and columns represent labels, label 0=False, 1=True, -1=undecided) for explainers based on four different ML models trained on training set of Census dataset. Predictions by a given models are chosen as ground-truth values.

was causing this problem. The missing rules were causing information loss for labels, and hence weaker results were inevitable. We share a full explanation for this finding in section 4.4.

4.3 DDQN Results

We have trained a DDQN on time-series data depicting the price history of a given stock. We have gathered the necessary data for a stock symbol from a library called yfinance. We have selected the stock symbol "GARAN.IS", one of the prominent stocks in the BIST. We have used an hourly timeframe with an input size of 3462 data points, each representing a day with Open, Close, High, Low values for a given stock on that day. The exact position size of 1000$ is enforced for each trade, and initial equity for the agent is given as 2000$.

The input layer contains the following feature attributes, price, Relative Strength Index (RSI), Moving Average Convergence/Divergence (MACD), Exponential Moving Average (EMA), and Simple Moving Average (SMA). These are the most known technical indicators. Except for the MACD, we have used window_size of 10 for the period in the indicator calculations. Out of many random trials, we have found the optimum parameters for our neural network (online in the context of DDQN) as the following input layer, which has five nodes representing the price and the indicators explained above. We have two hidden layers with 256 and 256 node sizes. Lastly, we have three nodes in the output layer representing the following states HOLD, BUY AND SELL. We have used ReLU [38] activation function at the end of each layer in our network except the last one. In the last hidden layer, we have used the softmax function [39] for gathering a vector of probabilities for each output label which sums to 1 so that we can select the maximum probability out of three states. The reason for choosing softmax is because of the scaling operation. Otherwise, our DDQN tends to produce bigger values each time which creates an inconsistency.

We share return profits in percentage for the model and the explainer and the similarity percentage between the model and the explainer. We can see from Figure 4.8, that as we increase the minimum required rule number, we see an increase in the accuracy

Table 4.8: ML model results on test set of the "GARAN.IS" stock price history

model_name	model_return	rule_return	similarity	rule_class_similarity	min_required_rule_number
DDQN	8.6%	53.4%	37.4	[0.3 0.1 0.4]	1
DDQN	8.6%	48.5%	44.0	[0.3 0.1 0.5]	3
DDQN	8.6%	48.6%	48.4	[0.3 0.1 0.6]	4

for class 2 (SELL). We have also observed that the number of rules for that label increased. However, as the number grows, the rule becomes difficult to read.

We have trained DDQN with different batch_size() and window_size values. Nevertheless, the results did not change too much. We have concluded that our explainer cannot give good results for a Deep Learning model. We believe that our rule set is too small to find the complexity of a DL model. As we increase the minimum required rule number, we see a slight improvement in the explainability. However, the rule set becomes unreadable after the number is selected as four. This conflicts with our initial goal, which is readability. On the other hand, we also know that DDQN is a much more complex model than other ML models used in this work. Consequently, we acknowledge that a complex model can be hard to interpret by simple if-then rules.

4.4 Effect of minimum required number of rule for a label

We examine the behaviour of our explainer when all of the rules are extracted until the minimum required number of rules exist in case of a removal of the rules due to intersection handling. We observe the changes in the metrics as we increase our minimum number of the rules.

We want to emphasize that we do not provide the confusion matrices in a visual format as we did in previous sections in this Chapter. Instead, we share only the diagonal values in tabular format. We believe the tabular format would be easier to follow since we want to show the increase in the numbers.

Starting from Table 4.9 to Table 4.10 and Table 4.11, we can see that the accuracy score for the class 0 (the first item of the list in the rule_class_accuracy column) increases each time. It should also be noted that the number of rules in each table for

Table 4.9: ML model results on test set of Census Income when minimum required number of rules for a label is 2

model_name	model_acc	rule_acc	similarity	model_class_accuracy	rule_class_accuracy	rule_class_similarity
DT	99.002	63.949	63.166	[0.999, 0.963]	[0.544, 0.935]	[0.538, 0.933]
KNN	86.442	60.326	59.542	[0.932, 0.657]	[0.492, 0.947]	[0.487, 0.996]
SVC	83.341	61.354	56.717	[0.943, 0.497]	[0.516, 0.914]	[0.484, 0.991]
LogReg	82.251	54.153	55.52	[0.929, 0.493]	[0.516, 0.62]	[0.489, 0.87]

Table 4.10: ML model results on test set of Census Income when minimum required number of rules for a label is 3

model_name	model_acc	rule_acc	similarity	model_class_accuracy	rule_class_accuracy	rule_class_similarity
DT	99.002	68.647	67.895	[0.999, 0.963]	[0.613, 0.912]	[0.607, 0.911]
KNN	86.442	68.263	69.261	[0.932, 0.657]	[0.616, 0.887]	[0.618, 0.97]
SVC	83.341	65.899	61.799	[0.943, 0.497]	[0.576, 0.914]	[0.544, 0.991]
LogReg	82.251	61.124	63.058	[0.929, 0.493]	[0.608, 0.62]	[0.58, 0.87]

class 0 is always equal to the minimum required number provided to the SHAPRule-Generator. This means that, for class 0 in each case, our explainer had already omitted the rules for the first class, which is the reason why the total number of rules increased when the minimum required number increases.

We can also see the increase in rule accuracy (rule_acc column) and similarity columns as well as similarity values for class 0 when the required number for rule increases. On the other hand, class 1 accuracies in the column rule_accuracy drop as the minimum required number increases. This situation occurs because of the introduced rules, which were omitted previously. Those rules stole some of the instances that would go into class 1. SHAP Rule Generator eliminates promising candidate rules due to intersections between other labels. This finding suggests that the intersection handling algorithm should be improved.

Table 4.11: ML model results on test set of Census Income when minimum required number of rules for a label is 4

model_name	model_acc	rule_acc	similarity	model_class_accuracy	rule_class_accuracy	rule_class_similarity
DT	99.002	78.044	78.151	[0.999, 0.963]	[0.804, 0.709]	[0.801, 0.719]
KNN	86.442	70.828	72.501	[0.932, 0.657]	[0.665, 0.842]	[0.669, 0.934]
SVC	83.341	75.296	79.088	[0.943, 0.497]	[0.767, 0.711]	[0.764, 0.928]
LogReg	82.251	75.818	81.038	[0.929, 0.493]	[0.782, 0.686]	[0.789, 0.911]

CHAPTER 5

CONCLUSIONS

In this thesis, we have proposed a new explainability approach by using an existing explainability technique called SHAP. We have demonstrated a workflow to interpret black-box models for classification tasks with the help of Decision Trees along with KernelSHAP. We have introduced the following novelties to the literature:

1. Use of Shapley Values (we have used an estimator which is SHAP [1]) for extraction of if-then based rule sets
2. Ready to run software code that can mimic the black-box ML model that is trained for classification tasks
3. Explainability of DDQN model trained on stock market data

Our work has exceeded expectations for simple datasets such as Iris and Heart Attack. Although it scored much lower in the more complex model DDQN, the SHAP Rule generator performed well for simple models and datasets. Furthermore, our work allows for better results with the adjustments of Hyperparameters such as the minimum required rule number, which we have seen the impact in the census dataset. Our method extracts rules as a group and tries to interpret the given model globally. We think that rules/nodes in the Decision Trees for SHAP can also be used individually. Thus better set of rules can be extracted for local explainability instead of generating a whole prediction method as we do. The drawback of our method is that the explainer has to select one of the rules in favor of another when an intersection occurs. Hence, information loss occurs and our accuracy scores drop. This reduces the coverage of a rule or even can delete the rule entirely. Even though a small intersection occurs

between two rules for different labels, our explainer makes them mutually exclusive. This can be bad if many instances fall into the same intersection. This is the price to pay for interpretability. Otherwise, a full explanation for complex models would be complicated. We believe a more intelligent intersection handling algorithm can be developed, which we leave as future work.

Our work is limited to classification tasks applied to tabular data. It is worth emphasizing that our work depends on the credibility of the SHAP library. When the dataset becomes quite huge, our explainer's runtime lasts longer because of compute-heavy calculations made by the SHAP library. Furthermore, our explainer may not produce readable outcomes when the feature size increases dramatically. This might introduce too many if-then rules, which are challenging to interpret. If there are dependencies between feature attributes, SHAP might perform poorly. If the SHAP performs poorly, it will also impact the explainer's result. In addition, finding the optimum number for the minimum required rule is greedy, which is not credible. More intelligent solutions can be designed for finding the best number in future work.

We give importance to reusability in our field. Therefore, we share our code repository in the following GitHub link as the basis of our work [1].

[1] https://github.com/egemenberk/shap_rule_generator

REFERENCES

[1] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[2] A. M, "Ddqn implementation for stock trading." `https://github.com/DeepNeuralAI/RL-DeepQLearning-Trading/blob/master/public/ddqn.svg`.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, ""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, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.

[6] Z. Obermeyer and E. J. Emanuel, "Predicting the future—big data, machine learning, and clinical medicine," *The New England journal of medicine*, vol. 375, no. 13, p. 1216, 2016.

[7] H. Kvamme, N. Sellereite, K. Aas, and S. Sjursen, "Predicting mortgage default using convolutional neural networks," *Expert Systems with Applications*, vol. 102, pp. 207–217, 2018.

[8] K. Aas, M. Jullum, and A. Løland, "Explaining individual predictions when fea-

tures are dependent: More accurate approximations to shapley values," *Artificial Intelligence*, vol. 298, p. 103502, 2021.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[10] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

[11] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[12] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[15] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Definitions, methods, and applications in interpretable machine learning," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 22071–22080, oct 2019.

[16] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[17] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *AAAI*, 2018.

[18] K. Aas, M. Jullum, and A. Løland, "Explaining individual predictions when features are dependent: More accurate approximations to shapley values," 2019.

[19] J. Christopher, "Watkins and peter dayan. q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.

[20] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[21] M. H. Akyuz and S. I. Birbil, "Discovering classification rules for interpretable learning with linear programming," 2021.

[22] W. W. Cohen and Y. Singer, "A simple, fast, and effictive rule learner," in *AAAI/IAAI*, 1999.

[23] Y. Shi, L. Wei, L. Zhu, K. Guo, and E. Cambria, "Stock trading rule discovery with double deep q-network," *Applied Soft Computing*, vol. 107, p. 107320, 03 2021.

[24] B. Liu, W. Hsu, Y. Ma, *et al.*, "Integrating classification and association rule mining.," in *Kdd*, vol. 98, pp. 80–86, 1998.

[25] G. Attanasio, L. Cagliero, and E. Baralis, "Leveraging the explainability of associative classifiers to support quantitative stock trading," in *Proceedings of the Sixth International Workshop on Data Science for Macro-Modeling*, pp. 1–6, 2020.

[26] J. H. Friedman and B. E. Popescu, "Predictive learning via rule ensembles," *The Annals of Applied Statistics*, vol. 2, sep 2008.

[27] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[28] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 673–692, 2004.

[29] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[30] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997.

[31] A. M, "Rl-deepqlearning-trading." https://github.com/DeepNeuralAI/RL-DeepQLearning-Trading#readme, 2020.

[32] R. Aroussi, "Download market data from yahoo! finance's api." https://github.com/ranaroussi/yfinance.

[33] J. Benediktsson, "Python wrapper for ta-lib." https://github.com/mrjbq7/ta-lib.

[34] kernc, "Backtest trading strategies in python.." https://kernc.github.io/backtesting.py/.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] R. Rahman, "A dataset for heart attack classification." https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset.

[37] R. Kohavi and B. Becker, "Data mining and visualization, silicon graphics," 1994. Available at https://archive.ics.uci.edu/ml/datasets/census+income.

[38] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[39] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*, pp. 227–236, Springer, 1990.

APPENDIX A

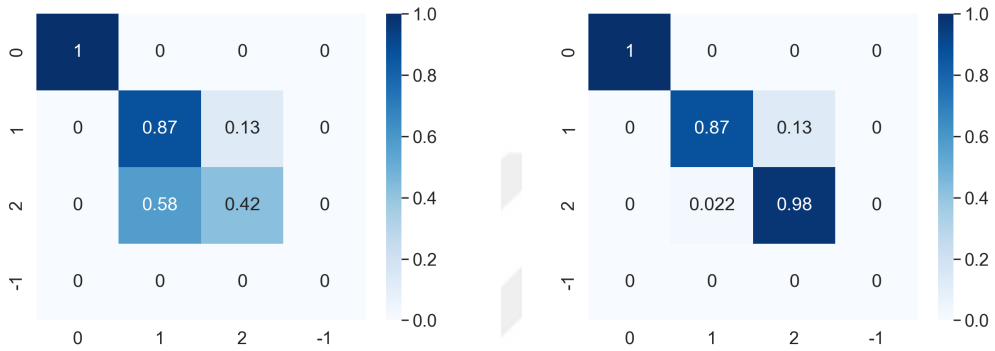
ACCURACY METRICS FOR ML MODELS AND THE EXPLAINER

A.1 Accuracy metrics for model and SHAP Rule Generator on Iris training dataset

In this section, we share the accuracy metrics for the ML models trained in the Chapter 4. The reason for showing these results is to verify that our ML models do not experience underfitting nor overfitting. In each figure, rows and columns represent the labels of the dataset. Sum of the each values in the same row equals to 1. Diagonal values represent the accuracy for model or the given explainer per each label.

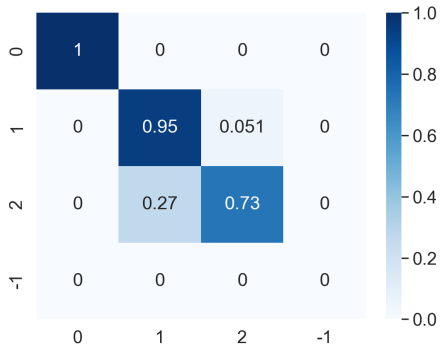
A.2 Accuracy metrics for model and SHAP Rule Generator on Heart Attack training dataset

A.3 Accuracy metrics for model and SHAP Rule Generator on Adult Census training dataset

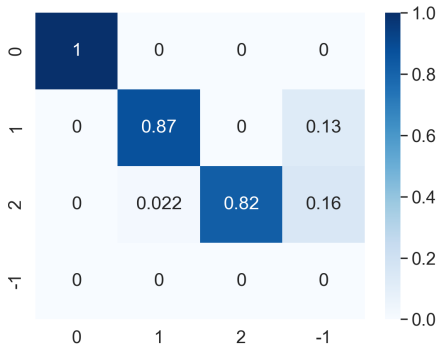


(a) SVC

(b) KNN

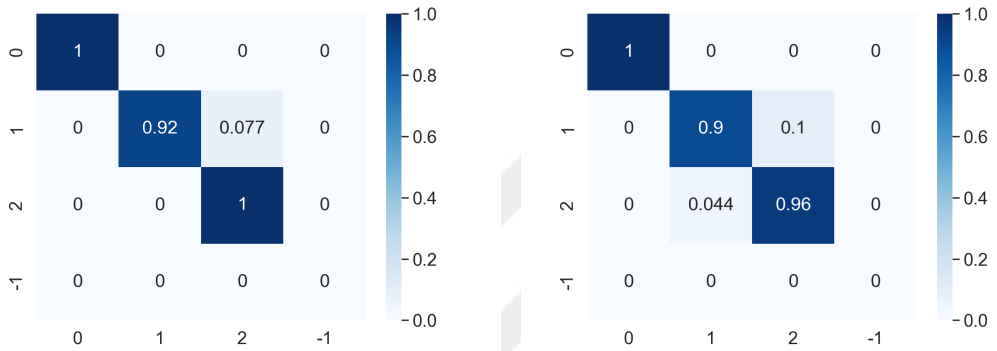


(c) DT



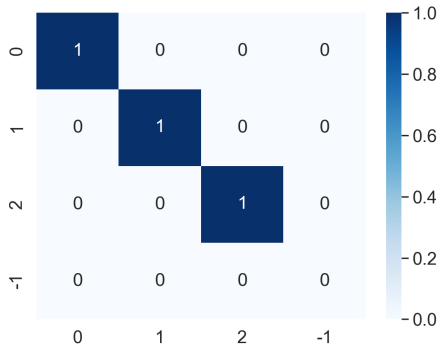
(d) LogReg

Figure A.1: Confusion Matrix for accuracy of explainers based on four different ML models trained on Iris training dataset.

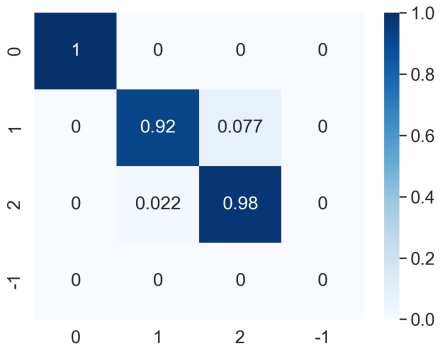


(a) SVC

(b) KNN



(c) DT



(d) LogReg

Figure A.2: Confusion Matrix for accuracy of four different ML models trained on Iris training dataset.

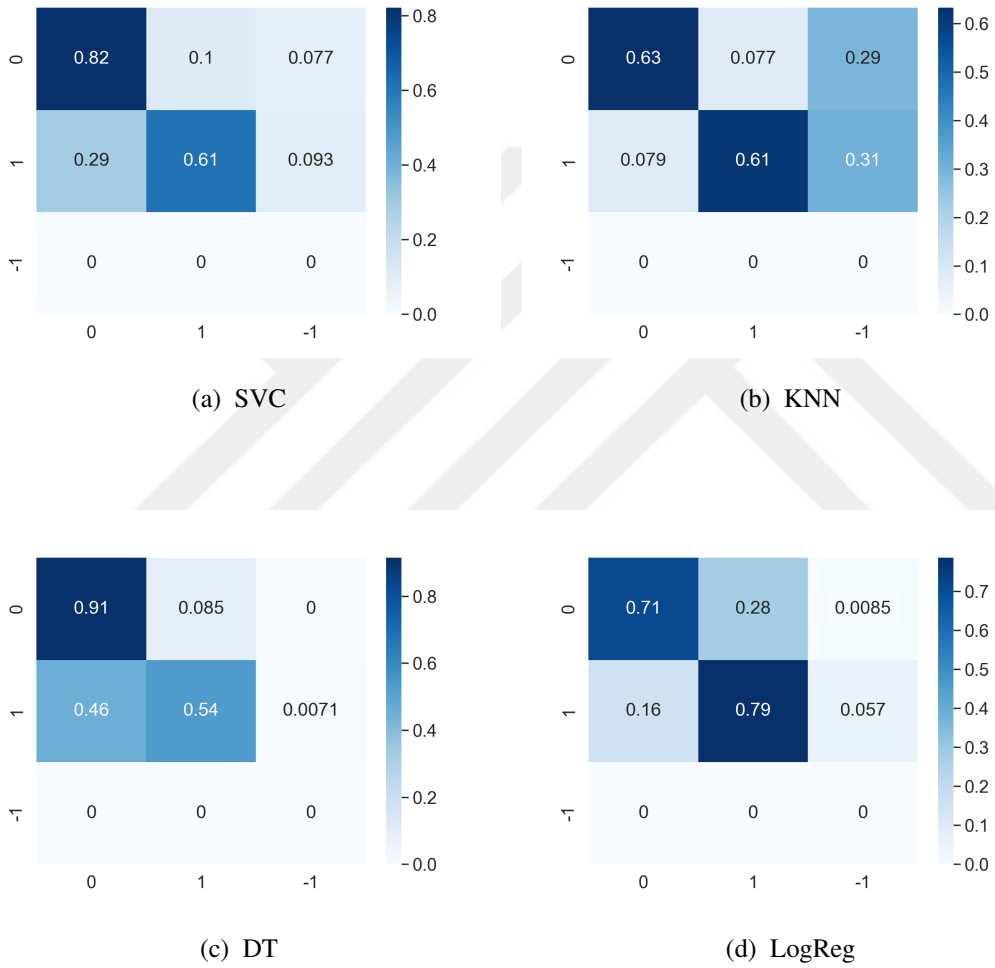
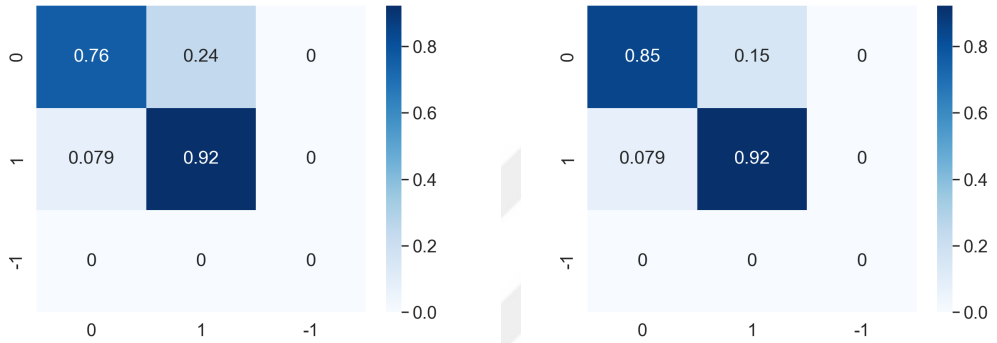
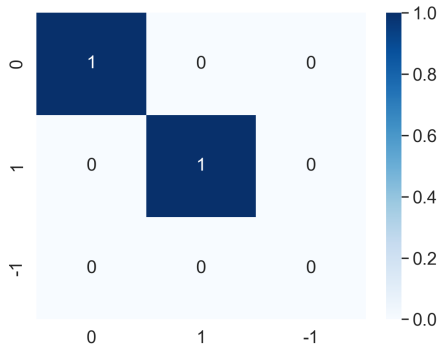


Figure A.3: Confusion Matrix for accuracy of explainers based on four different ML models trained on Heart Attack training dataset.

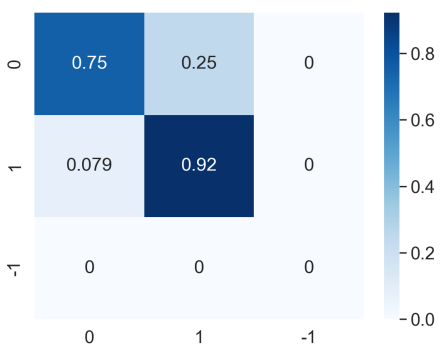


(a) SVC

(b) KNN

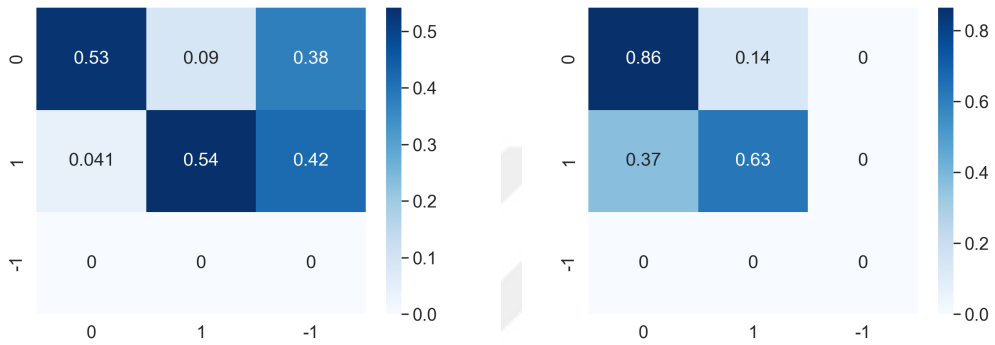


(c) DT



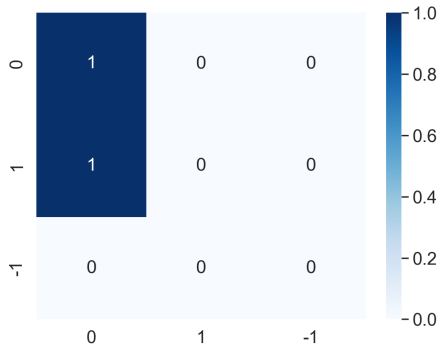
(d) LogReg

Figure A.4: Confusion Matrix for accuracy of four different ML models trained on Heart Attack training dataset.

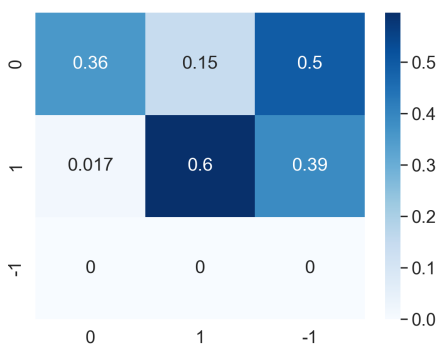


(a) SVC

(b) KNN

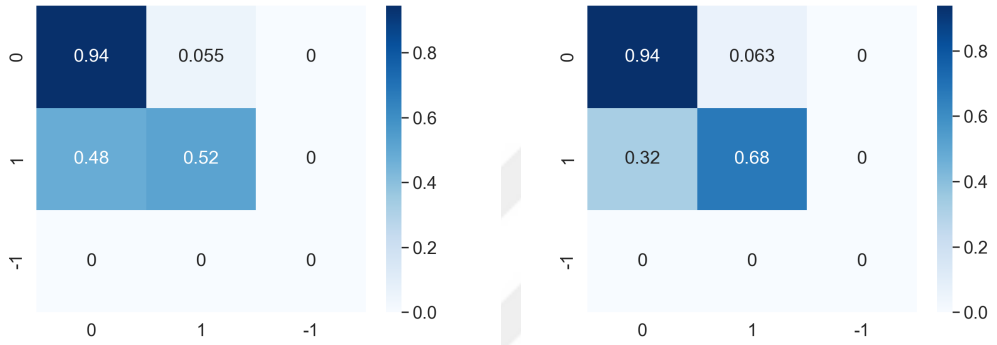


(c) DT



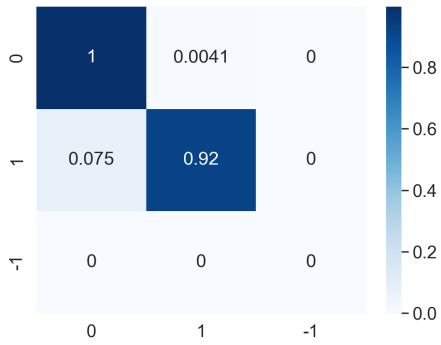
(d) LogReg

Figure A.5: Confusion Matrix for accuracy of explainers based on four different ML models trained on Adult Census training dataset.

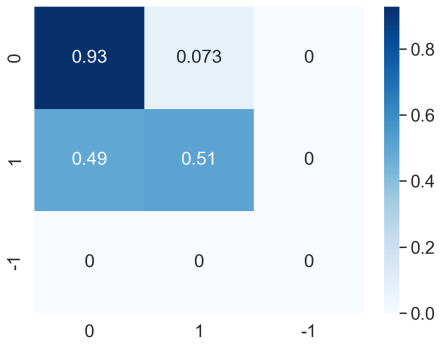


(a) SVC

(b) KNN



(c) DT



(d) LogReg

Figure A.6: Confusion Matrix for accuracy of four different ML models trained on Adult Census training dataset.