EVALUATION AND COMPARSION OF METHODS OF QUANTIFYING EXPLAINABILITY IN MACHINE LEARNING MODELS

A REPORT SUBMITTED TO
SAVITRIBAI PHULE PUNE UNIVERSITY
TOWARDS PARTIAL FULFILLMENT OF
DEGREE OF MASTER OF SCIENCE (M.Sc.)
IN STATISTICS
IN THE FACULTY OF SCIENCE AND TECHNOLOGY

Submitted By

MOHIT JADHAV (2118)
DIGVIJAY PATIL (2140)
ROMIT SURYAWANSHI (2152)

Under The Guidance of

DR. AKANKSHA KASHIKAR

DEPARTMENT OF STATISTICS

AND

CENTRE FOR ADVANCED STUDIES IN STATISTICS,

SAVITRIBAI PHULE PUNE UNIVERSITY,

PUNE-411007,

INDIA.

MAY, 2023

Certificate of the Guide

This is to certify that, the following students of M.Sc. Statistics,

(1) **Mohit Jadhav**

(2) **Digvijay Patil**

(3) **Romit Suryawanshi**

have successfully completed their project titled **Evaluation and Comparison of Methods of Quantifying Explainability in Machine Learning Models** under the guidance of **Dr.Akanksha Kashikar** and have submitted this project report on **22 May 2023** as a part of the course **ST-402**, towards partial fulfillment of requirements for degree of M.Sc. Statistics in Savitribai Phule Pune University in academic year 2022-2023.

Dr. Akanksha Kashikar	**Prof. T. V. Ramanathan**
(Project Guide)	(Head of the Department)

Acknowledgments

We would like to express my sincere gratitude to **Prof. TV Ramanathan**, Head of the Department, for his constant support and encouragement throughout this project. We are also grateful to **Dr.Akanksha Kashikar**, our project guide, for her valuable guidance, suggestions and feedback which helped us complete this project successfully. We would also like to thank my fellow colleagues for their cooperation and support throughout the project.

Contents

1 INTRODUCTION **1**

2 MOTIVATION **3**

3 APPLICATION **5**

4 LITERATURE REVIEW **7**

4.1 Ideas on interpreting machine learning. 7

4.2 Which Explanation Should I Choose? A Function Approximation Perspective to Characterizing Post Hoc Explanations by Han T, Srinivas S and Lakkaraju H (2022) 9

4.3 Comparing Explanation Methods for Traditional Machine Learning Models Part 1: An Overview of Current Methods and Quantifying Their Disagreement. Flora, Montgomery, et al(Nov 2022) 12

5 DATA DESCRIPTION **15**

6 TOOLS AND TECHNIQUES **19**

6.1 Black Box models 19

6.2 Explainers 21

6.3 Overview of actual outputs of explainers 23

7 ANALYSIS **27**

7.1 Evaluation 29

7.2 Comparison Using Different Matrics 34

8 CONCLUSIONS **37**

9 LIMITATIONS AND FUTURE SCOPE **41**

A APPENDIX **45**

A1 Acivation functions 45

A2 Metrics 47
A3 R Codes 49
A4 Python Code 53

Chapter 1

INTRODUCTION

Machine learning has become an invaluable tool for improving accuracy and automating decision-making in a wide range of fields, from finance and marketing to transportation and healthcare. The ability of these models to learn complex patterns from data and make predictions has led to impressive results, but in certain scenarios, their complexity has also raised concerns about their interpretability. In particular, in critical domains such as healthcare and criminal justice, the interpretability of machine learning models is of paramount importance. Suppose a healthcare provider is using a machine learning model to predict the likelihood of a patient developing a certain condition. The model has been trained on a large dataset and has achieved high accuracy. However, the healthcare provider needs to be able to understand how the model is making its predictions in order to make informed decisions about the patient's treatment.

The ability to understand how a model makes its predictions is crucial for decision-makers such as doctors, judges, and loan officers, as their decisions can have a direct and significant impact on people's lives. Therefore, the interpretability of machine learning models is becoming increasingly important as they are deployed in these domains. To address this challenge, several post-hoc explanation methods, or "explainers", have been developed to make the predictions of these models more transparent. An "explainer" refers to a set of tools or frameworks that can be used to help understand how a machine learning model works or makes decisions. These explainers have different goals and approaches, such as LIME (Local Interpretable Model-agnostic Explanations) which explains predictions locally by generating a linear model around the instance to be explained and C-LIME(Continuous-Local Interpretable Model agnostic Explanations), which uses clustering to generate counterfactual explanations. SHAP (SHapley Additive exPlanations), Vanilla gradient, Smooth Grad, and Integrated gradient are other examples of explainers, each of which use different techniques to explain the predictions of a model. These differences lead to conceptual and practical challenges in understanding and using explanation methods. This study aims to compare these explainers using simulated data for which we know the relation between predictors and response already, in order to determine their efficiency and suitability for different problem categories, such as regression and classification. By evaluating the performance of these explainers in terms of their ability to accurately and clearly explain the decisions made by the models, we aim to identify which explainers are best suited for different types of problems. Additionally, the study will investigate the performance of these explainers on real world datasets and provide insights on the limitations and strengths of each method.Overall, the goal of this study is to provide a comprehensive comparison of various explainers and help decision-makers in critical domains to understand and trust the predictions made by machine learning models. By providing guidance on which explainers to use in different situations, we aim to improve the interpretability and trustworthiness of machine learning models in these critical domains, as well as to provide a mathematical framework that unifies and characterizes different explainers and that provides approach to choose explainers.

■

Chapter 2

MOTIVATION

The motivation behind this project lies in addressing the need for reliable, and systematic evaluation and comparison of methods for quantifying explainability in machine learning models. By doing so, it aims to contribute to the advancement of the field, support decision-making, and promote trust, accountability, and ethical considerations in the application of machine learning models.

- **Context and Significance:** With the increasing adoption of machine learning models across industries, there is a growing demand for transparency and interpretability. It is crucial to understand the inner workings of these models and provide meaningful explanations for their decisions. This project aims to address the pressing need for reliable and effective methods to quantify explainability, considering the impact on decision-making and applications.

- **Diverse Range of Explainability Techniques:** There exists a wide range of methods proposed for quantifying the explainability of machine learning models. These include rule-based approaches, feature importance measures, model-agnostic methods, and post hoc explainers. However, there is a lack of consensus on which methods are most suitable for different scenarios. This project seeks to systematically evaluate and compare these methods, providing a comprehensive understanding of their strengths, limitations, and applicability.

- **Decision-Making Implications:** The explainability of machine learning models directly affects their deployment and usability in real-world applications. By quantifying the explainability of different methods, this project can offer valuable insights into the trade-offs between accuracy and interpretability. It aims to provide decision-makers with evidence-based guidance in selecting the most appropriate explainability techniques for their specific requirements and constraints.

- **Identifying Limitations and Advancements:** Through rigorous evaluation and comparison, this project will uncover the limitations and potential advancements in existing explainability methods. By identifying gaps in current techniques, it aims to contribute to the development of more effective and robust approaches for quantifying explainability. This will help researchers and practitioners stay informed about the state of the art and drive advancements in the field.

- **Trust, Accountability, and Ethical Considerations:** Explainability is crucial for building trust and ensuring accountability in machine learning models. Stakeholders, including regulators, users, and customers, increasingly demand explanations for the decisions made by these models. By providing a systematic evaluation and comparison of explainability methods, this project addresses ethical considerations and supports responsible deployment of machine learning models.

■

Chapter 3

APPLICATION

In this section, we have discussed about some applications of explainable AI.

- **Regulatory Compliance:** Industries such as finance, healthcare, and insurance are subject to regulatory frameworks that require explainability and transparency in decision-making. This project's findings can help organizations in these domains assess the suitability of different explainability methods for complying with regulations and providing justifications for their model outputs.

- **Model Validation and Selection:** When deploying machine learning models, organizations often need to choose between multiple models or variations of the same model. This project's evaluation and comparison of explainability methods can assist in assessing the interpretability and explainability of these models. This information can guide organizations in selecting the most appropriate model that aligns with their interpretability requirements.

- **Trust and User Acceptance:** Explainable machine learning models are more likely to be trusted and accepted by end-users. By quantifying the explainability of different methods, this project can provide insights into the comprehensibility and transparency of model decisions. This can enhance user trust and acceptance in various applications, including recommendation systems, fraud detection, and autonomous vehicles.

- **Model Improvement and Debugging:** The insights gained from evaluating and comparing explainability methods can aid in improving and debugging machine learning models. By identifying the variables or features that have the greatest impact on model predictions, organizations can refine their models, eliminate biases, and enhance overall performance.

- **Ethical Considerations:** This project can contribute to the ethical deployment of machine learning models by assessing the fairness, bias, and discriminatory aspects of different explainability methods. This knowledge can help organizations identify potential pitfalls and ensure the responsible use of AI technologies.

- **Educational and Research Purposes:** The outcomes of this project can serve as a valuable resource for researchers, practitioners, and educators in the field of machine learning explainability. The evaluation and comparison of methods can provide a foundation for further research and development of novel explainability techniques.

■

Chapter 4

LITERATURE REVIEW

4.1 Ideas on interpreting machine learning.

(a) Introduction : The article provides ideas on interpreting machine learning models. It suggests that traditional assessment practices may not be enough to understand how a model is making predictions, and that sensitivity analysis can be used to understand how small changes in input variable values can affect the model's predictions. The article suggests using techniques such as feature importance and partial dependence plots to understand how a model uses input variables and LIME and SHAP to understand individual predictions. It is an article discussing various methods for interpreting machine learning models and gaining insight into how they make predictions. The article provides an overview of the methods and how they can be used to understand the predictions of machine learning models, but it does not present any specific results or conclusions from apply- ing these methods to any particular model or dataset. It is intended to be a guide for practitioners who are interested in interpreting machine learning models and gaining a deeper understanding of how they make predictions.

(b) Methods and Techniques discussed: Explanation of some of the methods mentioned in the article:

Sensitivity analysis: This is a method used to understand how small changes in input variable values can affect the model's predictions. It can be used to identify which input variables have the greatest impact on the model's predictions, and how changing the values of these variables can affect the model's behaviour.

Feature importance: This is a technique used to understand how a model is using different input variables to make predictions. It can be used to identify which input variables have the greatest impact on the model's predictions. There are various ways to measure feature importance, such as permutation feature importance, mean decrease impurity and mean decrease accuracy. Permutation feature importance is a technique used to determine which features are most important in a machine learning model. It works by randomly shuffling or changing function values one at a time and seeing how that affects the performance of the model. If a model's performance drops significantly when a particular feature changes, that feature is considered more important. This method helps you understand which features are most important for a given model and also helps you identify potential problems with your model. (see `https://www.kaggle.com/code/dansbecker/permutation-importance`)

These methods can be used in combination to gain a deeper understanding of how a machine learning model makes predictions, and how different input variables affect the predictions. However, it's worth nothing that these methods require a good understanding of the problem and the model's architecture, and some of them may be computationally expensive.

4.2 Which Explanation Should I Choose? A Function Approximation Perspective to Characterizing Post Hoc Explanations by Han T, Srinivas S and Lakkaraju H (2022)

(a) Introduction: The objective of this paper is to address the challenges in post-hoc explainability by unifying eight popular explanation methods (LIME, C-LIME, SHAP,Occlusion, Vanilla Gradients, Gradients × Input, SmoothGrad, and Integrated Gradients).It also states that they all perform local function approximation of the black-box model, differing only in the neighborhood and loss function used to perform the approximation. The authors aim to provide a guiding principle for choosing among methods based on faithfulness to the black-box model and empirically validate their theoretical results using various real-world datasets, model classes, and prediction tasks. The paper aims to advance the conceptual understanding of these methods and guide their use in practice, providing a principled approach to choosing among methods.

(b) Data Used: The paper has two real-world datasets for two prediction tasks. The first dataset is the life expectancy dataset from the World Health Organization (WHO) which contains demographic, economic, and health information of countries from 2000 to 2015, with 2,938 observations and 20 continuous features. They use this dataset to perform regression and predict life expectancy. The second dataset is the home equity line of credit (HELOC) dataset from FICO (Fair Isaac Corporation-a data analytics company), consisting of information on HELOC applications, with 9,871 observations and 24 continuous features. They use this dataset to perform classification and predict whether an applicant made payments without being 90 days overdue.

(c) Methods Used: In this paper, four models are trained .They train a linear regression model and three neural networks with 8-node hidden layers and tanh activation for the hidden layers and linear activation for the output layer. The neural networks have 3, 5, and 8 hidden layers and will be referred to as NN1, NN2, and NN3 respectively. For the HELOC dataset, they train a logistic regression model and three neural networks with 8-node hidden layers and ReLU (Rectified Linear Unit) activation for the hidden layers and sigmoid activation for the output layer.The first neural network had 3 hidden layers and was named NNA, the second neural network had 5 hidden layers and was named NNB, and the third neural network had 8 hidden layers and was named NNC. All models were trained using an 80/20 train/test split with stochastic gradient descent and decent model performance was considered more important than fine-tuning the hyperparameters. the linear and logistic regression models, the training and evaluation process was repeated 100 times with different random splits of the data into training and testing sets. Each repetition is known as a "fold" in the cross-validation process. On the other hand, for the neural network models, the training and evaluation process was repeated 300 times with different random splits of the data into training and testing sets. All models used a batch size of 64, i.e 64 samples were used at each step to update the model parameters during training of the optimization algorithm. Cosine annealing scheduler was used to adjust the learning rate over the course of training. The "learning rate" is a hyperparameter that controls the step size taken in the direction of the negative gradient during optimization. A smaller learning rate can result in slower convergence to the optimal solution, but a larger learning rate can lead to overshooting and instability in the optimization pro- cess. Cosine annealing scheduler gradually reduces the learning rate from an initial value to a minimum value and then increases it again in a cosine pattern. The cosine annealing scheduler can help the optimization algorithm to escape from local minima and converge to a better global minimum.

(d) Results: The results obtained from the paper include the formalization of the local function approximation (LFA) framework and the demonstration that eight popular explanation methods can be characterized as instances of this framework with different local neighbourhoods and loss functions. The authors also introduced the "no free lunch theorem" for explanation methods, showing that no single method can perform optimally across all neighbourhoods and provided a guiding principle for choosing among methods. The

LFA framework brings conceptual coherence and clarity to di- verse explanation methods, and enables theoretical simplicity when studying them.

4.3 Comparing Explanation Methods for Traditional Machine Learning Models Part 1: An Overview of Current Methods and Quantifying Their Disagreement. Flora, Montgomery, et al(Nov 2022)

(a) Introduction : The paper discusses interpretability and explainability in machine learning models, particularly for high-risk applications like severe weather forecasting. It distinguishes between interpretability (understandability without additional methods) and explainability (approximate understanding through post-hoc methods). The authors summarize various explainability methods and acknowledge the disagreements between them. They propose quantitative measures to assess agreement among methods and provide practical advice for interpreting conflicting explanations. The paper aims to promote further research on evaluation criteria for explanation methods.
(b) Data used: The severe weather dataset, derived from the experimental Warn-on-Forecast System (WoFS) and includes features extracted from ensemble storm tracks and environmental variables. The road surface dataset spans two cool seasons and contains features, including near-surface variables from the High Resolution Rapid Refresh (HRRR) model, used to predict road surface temperatures.
(c)Methods Used : This study uses classification random forests and logistic regression models. Logistic regression with elastic nets is employed for classification, while random forests are used for predictive modeling. The study focuses on global explainability, and local explainability. Explanation methods include feature importance, feature effects, and feature interactions. Multiple explanation methods are employed, as each has its strengths.
Global Explainability Methods used :
Feature Importance vs. Relevance: Feature ranking methods can be categorized into three types:
a) Single-pass and multi-pass permutation importance: Quick to compute,

model-agnostic, but sensitive to correlated features.
b) Gini Impurity: Based on decision tree or forest impurity score, computed at training time, but sensitive to correlated features.
c) Shapley Additive Global Importance (SAGE): Uses modified Shapley theory, model-agnostic, but limited to certain methods and assumes independent features.
d) Partial Dependence (PD) and Accumulated Local Effects (ALE): Measure global model sensitivity to a feature, quick to compute, but sensitive to correlated features.
<u>Permutation Importance Methods:</u>
a) Single-pass: Measures univariate contribution of features to model performance.
b) Multi-pass: Measures multivariate contribution, accounts for feature codependencies.
c) Grouped Permutation Importance: Highlights contribution of feature groups, reduces computational time.
These methods help in understanding feature relevance, importance, and their impact on model performance. They provide insights into feature interactions, correlations, and overall model behaviour.
<u>Local Explainability Methods used :</u>
Three common feature attribution methods for explaining individual predictions are:
Tree Interpreter: This method analyzes changes in the proportion of positive examples in different regions of the feature space defined by a decision tree.
LIME: LIME fits a weighted linear regression model on perturbed data in the neighborhood of the example to be explained.
Shapely-based Methods: Shapley values quantify the contribution of each feature by considering its inclusion or exclusion in feature subsets. Owen values handle correlated features by grouping them into coalitions. These methods provide insights into the contributions of individual features to model predictions.
(d) Results: Different feature ranking methods can provide varying results,

leading to disagreements in the importance of features. However, each ranking is plausible, and relying on a single explanation can lead to confirmation bias. The agreement between methods can be measured using top feature agreement and feature rank agreement statistics. While there is some consensus among the methods, there are also differences, particularly when features are correlated. The inclusion of poorly performing methods does not significantly inflate disagreement. Feature interactions and correlations can impact rankings, and certain methods handle them better than others. Models with stronger feature interactions tend to have less rank agreement between methods. Grouped permutation importance can highlight the contribution of feature groups. In the WoFS dataset, intrastorm features are most important, while environmental and storm morphology features also play a role. For the road surface dataset, temperature-based features are more valuable than radiation-based features. Feature effects estimated by different methods generally align with the conditional event rate, indicating that the models are learning appropriate relationships. However, correlated features can lead to opposite effects in regression models. ALE and SHAP tend to agree well with each other and the event rate, while PD may underpredict feature effects. LIME shows varying correspondence with other methods and the event rate. SHAP demonstrates the potential for scaling from local to global explanations. ■

Chapter 5

DATA DESCRIPTION

To facilitate a comprehensive comparison of these explainable techniques, it is crucial to determine the precise relationship between the response variable and independent variables. This means identifying which regressors are genuinely significant in relation to the response variable. By doing so, we can ascertain the exact number of important variables that are correctly identified as significant by each corresponding explainer. To overcome this limitation and enable more robust comparisons and detailed analyses, we have generated simulated data using the following six equations.

$$y = 3x_1 + 2x_2 + 5x_3 - 4x_4 + 0.5x_5 + \sin(x_6) - \exp(x_7) + \epsilon \tag{5.0.1}$$

$$y = 2x_1 + 3log(x_2) + 4x_3^2 - 8\sqrt{x_4} + cos(x_5) + 0.3x_6 + \epsilon \tag{5.0.2}$$

$$y = x_1 + x_2 + \sqrt{x_3} + log(x_4) + exp(x_5) + 2tan(x_6) + 0.3x_1x_2 + x_7 + \epsilon \tag{5.0.3}$$

where

$$x_7 = \begin{cases} 0, & \text{if } x_5 \leq 0 \\ 1, & \text{otherwise} \end{cases}$$

$$y = 4x_1 + 8x_2^2 + 0.7x_3^3 + 5x_4^4 + 3x_1x_2 + 10x_2x_3 - 27x_3x_4 + 2x_1x_2x_3x_4 + \epsilon \tag{5.0.4}$$

$$y = 4x_1 + 3x_2^2 + 45x_3^5 + log(x_1) + \frac{1}{x_3} + \epsilon \tag{5.0.5}$$

where

$$y = \begin{cases} 0, & \text{if } y \leq 0 \\ 1, & \text{if } y \geq 1 \end{cases}$$

$$z = 0.5x_1 + 0.8x_2 - 1.2x_3 + 3x_1 - 2x_1x_2 + x_5^2$$
$$p = 1/(1 + exp(-z)) \tag{5.0.6}$$

$$\text{y}\sim \text{ bernoulli(p)}$$

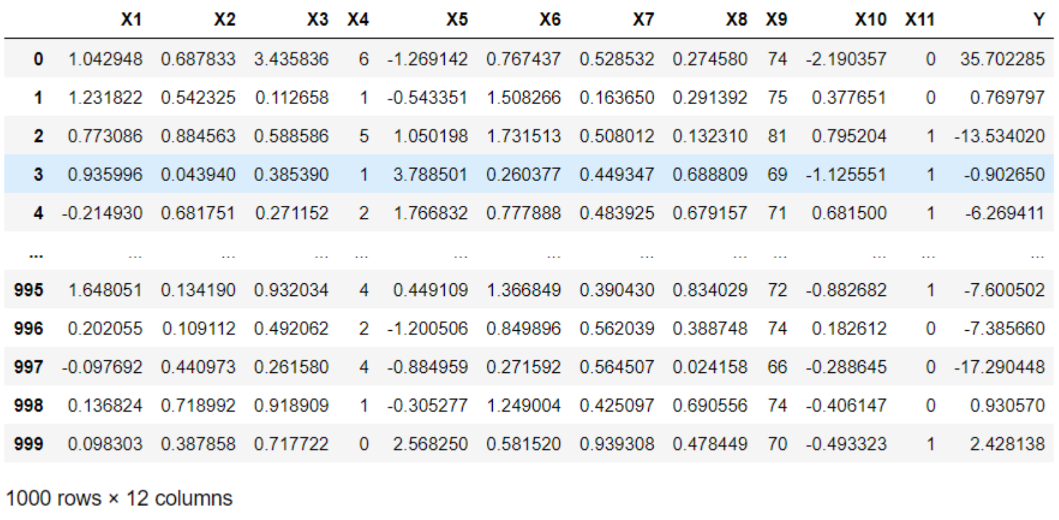
Here,

- x_1 is generated from Standard Normal distribution.
- x_2 is generated from Gamma distribution with shape parameter 1 and rate parameter 2.
- x_3 is generated from Exponential distribution with rate parameter of 1 .
- x_4 is generated from Poisson distribution having mean 2.5 .
- x_5 is generated from Normal distribution with mean 1 and standard deviation 2.
- x_6 is generated from Gamma distribution with shape parameter 3 and rate parameter 4.
- x_7 is generated from Standard Uniform distribution.
- x_8 is generated from Exponential distribution with rate parameter of 3.
- x_9 is generated from Binomial distribution of size=100 and probability of 0.7
- x_{10} or ϵ is generated from Standard Normal distribution. and
- x_{11} is binary takes 0 if x_5 is less than 0 and 1 if x_5 is greater than 0.

We tried to include a different range of variables in these datasets. Prior to model fitting, we introduced additional variables beyond those used in the equations. Consequently, we have knowledge of which variables are significant and which ones are not, relative to the given dataset. This insight greatly aids in evaluating the explainers.

The incorporation of simulated data offers a more comprehensive understanding of the performance of different explanation methods. It allows us to examine the subtleties and nuances of various explanation methods, gaining a deeper comprehension of their strengths and limitations. Moreover, the use of simulated data provides valuable insights into the level of agreement or disagreement among different explanation methods for specific instances. This facilitates a better assessment of the consistency of various explanation methods and helps identify areas where further improvements or refinements may be necessary.

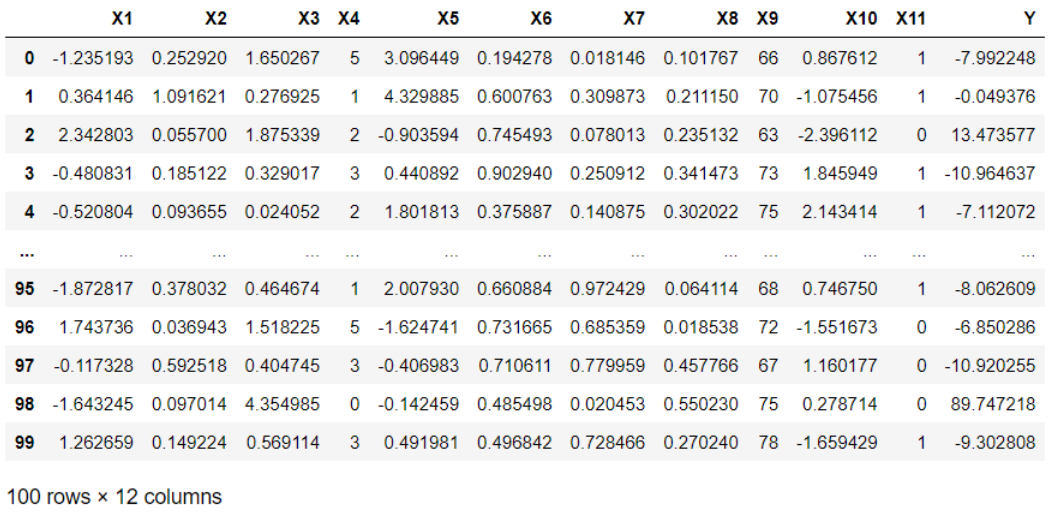
We have generated 1000 simulations using the above equations for different sizes of observations. i.e. we have 03 sets of 1000 simulations in which 1st set has observation size equals to 30, 2nd set has observation size equals to 100 , and 3rd set has observation size equals to 1000.
Figures given below are of the First simulation from First Equation with different observation sizes:



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	1.042948	0.687833	3.435836	6	-1.269142	0.767437	0.528532	0.274580	74	-2.190357	0	35.702285
1	1.231822	0.542325	0.112658	1	-0.543351	1.508266	0.163650	0.291392	75	0.377651	0	0.769797
2	0.773086	0.884563	0.588586	5	1.050198	1.731513	0.508012	0.132310	81	0.795204	1	-13.534020
3	0.935996	0.043940	0.385390	1	3.788501	0.260377	0.449347	0.688809	69	-1.125551	1	-0.902650
4	-0.214930	0.681751	0.271152	2	1.766832	0.777888	0.483925	0.679157	71	0.681500	1	-6.269411
...	...	...	...	...	...	...	...	...	...	...	...	...
995	1.648051	0.134190	0.932034	4	0.449109	1.366849	0.390430	0.834029	72	-0.882682	1	-7.600502
996	0.202055	0.109112	0.492062	2	-1.200506	0.849896	0.562039	0.388748	74	0.182612	0	-7.385660
997	-0.097692	0.440973	0.261580	4	-0.884959	0.271592	0.564507	0.024158	66	-0.288645	0	-17.290448
998	0.136824	0.718992	0.918909	1	-0.305277	1.249004	0.425097	0.690556	74	-0.406147	0	0.930570
999	0.098303	0.387858	0.717722	0	2.568250	0.581520	0.939308	0.478449	70	-0.493323	1	2.428138

1000 rows × 12 columns

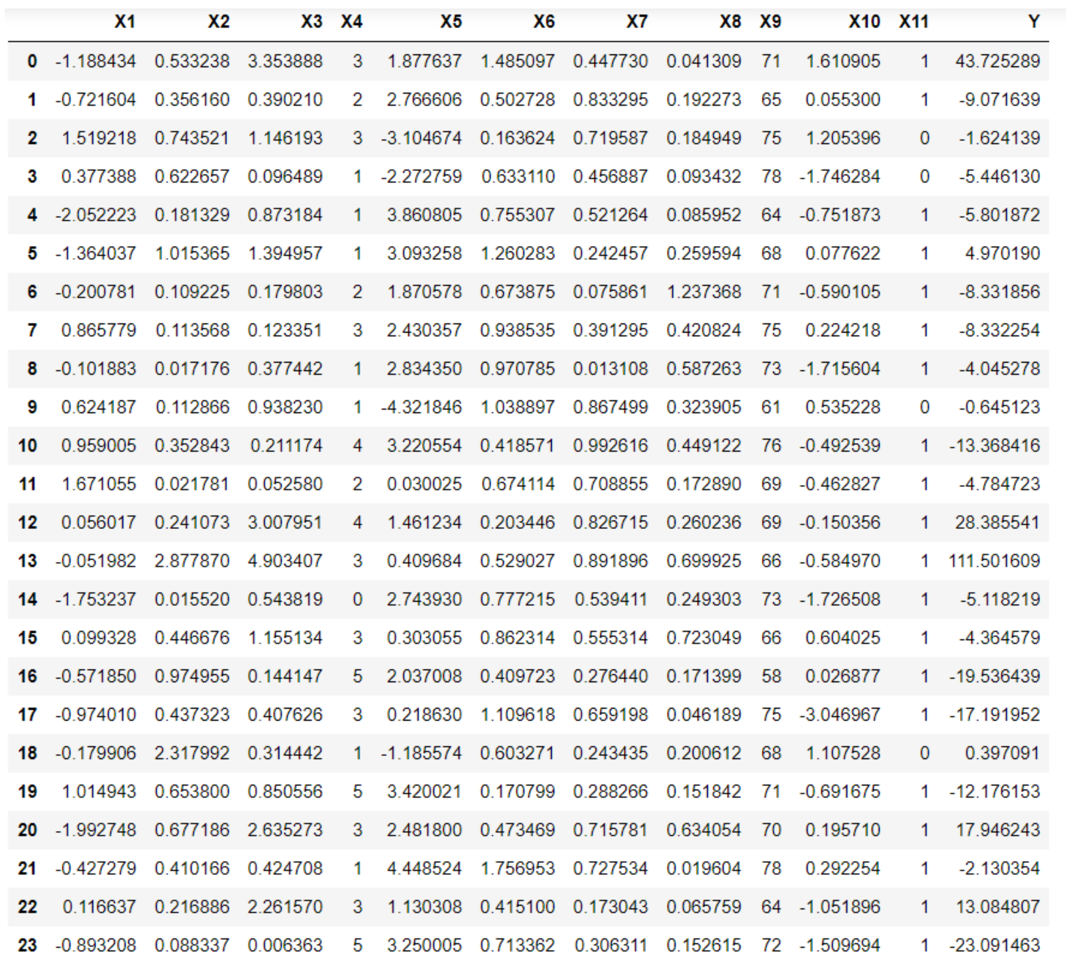
Figure 5.1: observation size equal to 1000



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	-1.235193	0.252920	1.650267	5	3.096449	0.194278	0.018146	0.101767	66	0.867612	1	-7.992248
1	0.364146	1.091621	0.276925	1	4.329885	0.600763	0.309873	0.211150	70	-1.075456	1	-0.049376
2	2.342803	0.055700	1.875339	2	-0.903594	0.745493	0.078013	0.235132	63	-2.396112	0	13.473577
3	-0.480831	0.185122	0.329017	3	0.440892	0.902940	0.250912	0.341473	73	1.845949	1	-10.964637
4	-0.520804	0.093655	0.024052	2	1.801813	0.375887	0.140875	0.302022	75	2.143414	1	-7.112072
...	...	...	...	...	...	...	...	...	...	...	...	...
95	-1.872817	0.378032	0.464674	1	2.007930	0.660884	0.972429	0.064114	68	0.746750	1	-8.062609
96	1.743736	0.036943	1.518225	5	-1.624741	0.731665	0.685359	0.018538	72	-1.551673	0	-6.850286
97	-0.117328	0.592518	0.404745	3	-0.406983	0.710611	0.779959	0.457766	67	1.160177	0	-10.920255
98	-1.643245	0.097014	4.354985	0	-0.142459	0.485498	0.020453	0.550230	75	0.278714	0	89.747218
99	1.262659	0.149224	0.569114	3	0.491981	0.496842	0.728466	0.270240	78	-1.659429	1	-9.302808

100 rows × 12 columns

Figure 5.2: observation size equal to 100



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	-1.188434	0.533238	3.353888	3	1.877637	1.485097	0.447730	0.041309	71	1.610905	1	43.725289
1	-0.721604	0.356160	0.390210	2	2.766606	0.502728	0.833295	0.192273	65	0.055300	1	-9.071639
2	1.519218	0.743521	1.146193	3	-3.104674	0.163624	0.719587	0.184949	75	1.205396	0	-1.624139
3	0.377388	0.622657	0.096489	1	-2.272759	0.633110	0.456887	0.093432	78	-1.746284	0	-5.446130
4	-2.052223	0.181329	0.873184	1	3.860805	0.755307	0.521264	0.085952	64	-0.751873	1	-5.801872
5	-1.364037	1.015365	1.394957	1	3.093258	1.260283	0.242457	0.259594	68	0.077622	1	4.970190
6	-0.200781	0.109225	0.179803	2	1.870578	0.673875	0.075861	1.237368	71	-0.590105	1	-8.331856
7	0.865779	0.113568	0.123351	3	2.430357	0.938535	0.391295	0.420824	75	0.224218	1	-8.332254
8	-0.101883	0.017176	0.377442	1	2.834350	0.970785	0.013108	0.587263	73	-1.715604	1	-4.045278
9	0.624187	0.112866	0.938230	1	-4.321846	1.038897	0.867499	0.323905	61	0.535228	0	-0.645123
10	0.959005	0.352843	0.211174	4	3.220554	0.418571	0.992616	0.449122	76	-0.492539	1	-13.368416
11	1.671055	0.021781	0.052580	2	0.030025	0.674114	0.708855	0.172890	69	-0.462827	1	-4.784723
12	0.056017	0.241073	3.007951	4	1.461234	0.203446	0.826715	0.260236	69	-0.150356	1	28.385541
13	-0.051982	2.877870	4.903407	3	0.409684	0.529027	0.891896	0.699925	66	-0.584970	1	111.501609
14	-1.753237	0.015520	0.543819	0	2.743930	0.777215	0.539411	0.249303	73	-1.726508	1	-5.118219
15	0.099328	0.446676	1.155134	3	0.303055	0.862314	0.555314	0.723049	66	0.604025	1	-4.364579
16	-0.571850	0.974955	0.144147	5	2.037008	0.409723	0.276440	0.171399	58	0.026877	1	-19.536439
17	-0.974010	0.437323	0.407626	3	0.218630	1.109618	0.659198	0.046189	75	-3.046967	1	-17.191952
18	-0.179906	2.317992	0.314442	1	-1.185574	0.603271	0.243435	0.200612	68	1.107528	0	0.397091
19	1.014943	0.653800	0.850556	5	3.420021	0.170799	0.288266	0.151842	71	-0.691675	1	-12.176153
20	-1.992748	0.677186	2.635273	3	2.481800	0.473469	0.715781	0.634054	70	0.195710	1	17.946243
21	-0.427279	0.410166	0.424708	1	4.448524	1.756953	0.727534	0.019604	78	0.292254	1	-2.130354
22	0.116637	0.216886	2.261570	3	1.130308	0.415100	0.173043	0.065759	64	-1.051896	1	13.084807
23	-0.893208	0.088337	0.006363	5	3.250005	0.713362	0.306311	0.152615	72	-1.509694	1	-23.091463

Figure 5.3: observation size equal to 30

Chapter 6

TOOLS AND TECHNIQUES

6.1 Black Box models

A black box model is a type of machine learning model that is treated as a "black box" because it is difficult or impossible to understand how it arrived at its decisions. This can be because the model is too complex, such as deep neural networks. These models can be very accurate in their predictions but can be hard to interpret and understand how they arrived at those predictions. Here is a list of examples of types of machine learning models that are considered to be black box models:

- Artificial Neural Networks (Deep Learning),
- Random Forest,
- Support Vector Machine,
- Gradient Boosting Machines (GBM),
- Extreme Gradient Boosting (XGBoost).

Note that this models are not exhaustive and the concept of black-box models are not only limited to these models. Some models can be considered as a black-box because of the complexity or lack of transparency in the model's output.

6.1.1 Black Box Models that We Used For Our Analysis

Artificial Neural Networks (Deep Learning)

An artificial neural network (ANN) is a type of machine learning model that is inspired by the structure and function of the human brain. It consists of layers of interconnected "neurons", which process and transmit information. The input data is fed into the input layer, and it is then passed through multiple hidden layers before reaching the output layer, where the final output is generated. Each neuron in a layer receives input from the previous layer, performs a computation on it, and passes the result on to the next layer. The computations performed by the neurons are determined by a set of parameters, called weights, which are learned by the model during training. ANNs are commonly used for tasks such as image recognition, natural language processing, and prediction.[Kamath and Liu, 2021]

Random Forest

This is an ensemble learning method that constructs a large number of decision trees on bootstrapped training samples, and then combines their predictions during inference. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from full set of predictors. The predictions of individual trees are combined through majority voting for classification, or averaging for regression.[Breiman, 2001]

Extreme Gradient Boosting (XGBoost)

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems. (see `https://www.geeksforgeeks.org/xgboost/`) [Chen and Guestrin, 2016]

6.2 Explainers

6.2.1 Explainers On which we have worked

LIME

Local Interpretable Model Agnostic Explanations (LIME) is a method for generating interpretable explanations of individual predictions made by any machine learning model. This approach is model agnostic, meaning it can be applied to any model, regardless of its training algorithm.

LIME is based on several key principles:

- Using interpretable representations of data that are easily understandable to humans
- Approximating the black-box model in the neighborhood of a data point with a linear, interpretable model
- Perturbing the data of interest to generate new samples, weighted by their proximity to the instance of interest
- Fitting the interpretable model to the new samples and using it to provide explanations of the black-box model

Formally, the problem of identifying a local surrogate model is equivalent to optimizing the objective function $\zeta(x)$ which is defined as:

$$\zeta(x) = \arg\min_{h \in \mathscr{H}} \{\mathscr{L}(f, h, \pi_x(z)) + \Omega(h)\}$$

where π_x is a kernel function that weighs a sample z based on its distance to the instance of interest x, $\mathscr{L}$ is a loss function, and Ω is a complexity penalty.

The explanation model, h is chosen from a class of interpretable models $\mathscr{H}$, such as linear, logistic, or decision tree models. The complexity term, $\Omega(h)$, is used to limit the number of model parameters or tree depth, by including a cost for increased complexity. The loss function serves as a metric of how inadequately the explanation model approximates the black-box model in the proximity of the instance x.[Ribeiro et al., 2016]

SHAP

Shapley values are a mathematical method for measuring feature importance in machine learning models. They are based on cooperative game theory and provide a fair way to distribute importance among features based on their contribution to the prediction. They have useful properties such as local accuracy and consistency, but do not handle feature interactions well and can be computationally expensive with a large feature space.

SHAP (Shapley Additive Explanations) is a method that explains a particular data instance as a unified measure of feature importance, providing a unique, additive feature importance explanation for individual data instances. It calculates Shapley values of a conditional expectation function of the model over a dataset.

$$f(x) = \mathbf{E}[f(X)|S]$$

Where S is a all possible subsets of a given set of features. However, as it is expensive to retrain a large set of models on subsets of features, SHAP applies the observed marginal expectation to calculate model predictions.

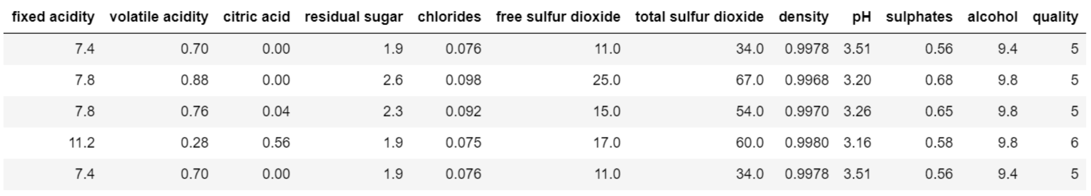
$$f(S) = \mathbf{E}[f(X)|S = S^*]$$

Where S^* are the feature values of data to be explained. Expectation is taken over a background dataset and several methods have been proposed to approximate the calculation of SHAP values.[Lundberg and Lee, 2017]

6.3 Overview of actual outputs of explainers

Before proceeding to actual comparison and use of the explainers to explain the predictions provided by underlying black box model. We have use these explainers on the toy data set to see how these work and to study the mechanism of different explainers. To gain deeper insights on how these explain model predictions for real life dataset we need to understand it's working and how to interpret their outputs. So in this section we have used the above mentioned explainers.

We will use the wine quality dataset. The goal is to predict the quality of wine scaled from 4-9 based on various features such as fixed acidity, volatile acidity, citric acid, residual sugar, density, pH, alcohol etc. We fitted black box model random forest to predict wine quality. Data we have used is
>wine.head()



fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

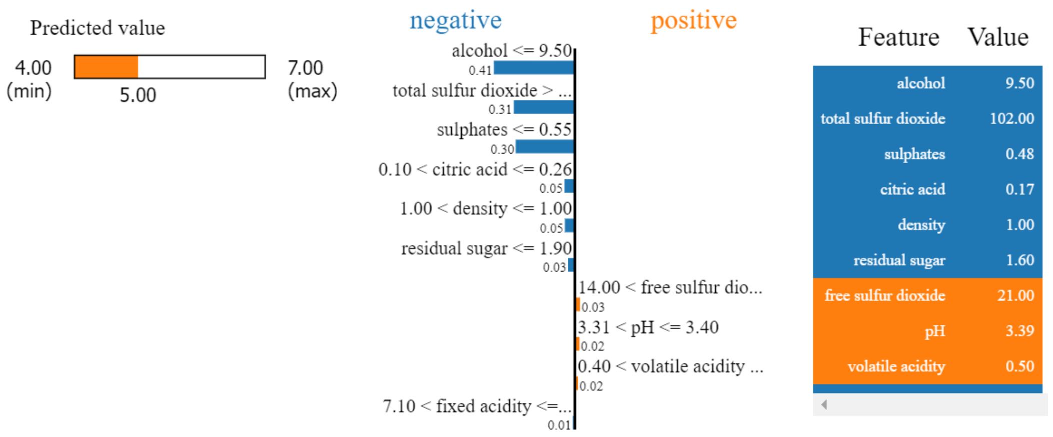
Figure 6.1: Wine quality

6.3.1 LIME

Outputs obtained from the LIME explainer are as follows

The above output shows three graphs that each show essential information about wine and its quality.

- The left graph shows that predicted value for above observation is 5.00.
- The center graph shows the feature importance scores on this particular sample. The length of the bar indicates the magnitude (absolute value), while the color indicates the sign (blue for negative, orange for positive) of the estimated coefficient.



Feature	Value
alcohol	9.50
total sulfur dioxide	102.00
sulphates	0.48
citric acid	0.17
density	1.00
residual sugar	1.60
free sulfur dioxide	21.00
pH	3.39
volatile acidity	0.50

Figure 6.2: LIME Explanation

- The right graph shows the actual values of top 9 important variables. Where, variables which are highlighted by blue colour, are the variables that pushing the response value towards 4.00 and variables which are highlighted by orange colour, are the variables that pushing the response value towards 7.00 for this particular observation.

6.3.2 SHAP

Outputs obtained from the SHAP explainer are as follows

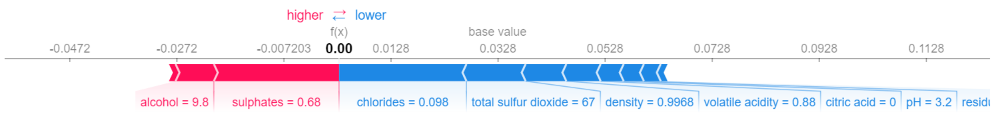


Figure 6.3: SHAP Explanation

- Let us break this elegant plot in great detail:
  The output value is the prediction for that observation (the prediction of the first row in our dataset is 0.

- **The base value is 0.0328 :** The value that would be predicted if we did not know any features for the current output. In other words, it is the mean prediction, or mean(yhat)

- **Red/blue:** Features that push the prediction higher (to the right) are shown in red, and those pushing the prediction lower are in blue.

- **Alcohol and Sulphates:** has a positive impact on the quality rating. The alcohol content of this wine is 9.8 which is higher than the average value and the Sulphates content of this wine is 0.68, which is higher than the average value and pushes the prediction to the right.

- Chlorides, total sulfur dioxide, density, volatile acidity,citric acid, pH, and residual sugar is positively related to the Wine quality. A lower than the average pushes the prediction to the left.

■

Chapter 7

ANALYSIS

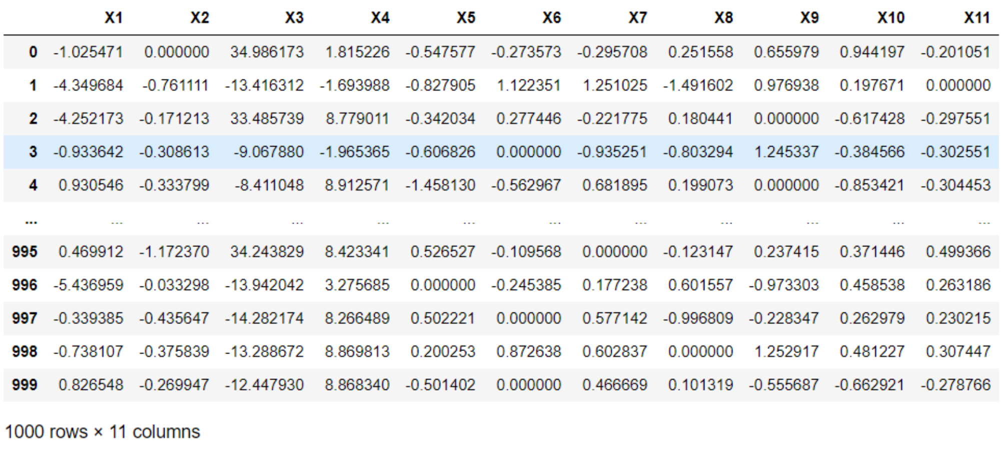
We experiment with 6 datasets for 2 prediction tasks. All the 6 datasets are generated using the equations described in Data Description. These datasets contains 11 variables out of which some are actually used to calculate response variable in corresponding equations and remaining are redundant variables. We used first 04 datasets to perform regression, predicting response. And remaining 2 datasets are used to perform classification, predicting whether a y=1 or y=0. For each dataset we have trained three models: a random forest, XGBoost and Artificial Neural Network.The Models were trained on 80/20 train/test split. For first four datasets, we have fitted three models

- Random Forest Regressor
- XGboost Regressor
- Feed-forword neural network

The neural network has 2 hidden layers with 'tanh' activation and an output layer with 'linear' activation.It is trained for 15 epochs and used a batch size of 32. For remaining datasets, we have fitted three models

- Random Forest Classifier
- XGboost Classifier
- Feed-forword neural network

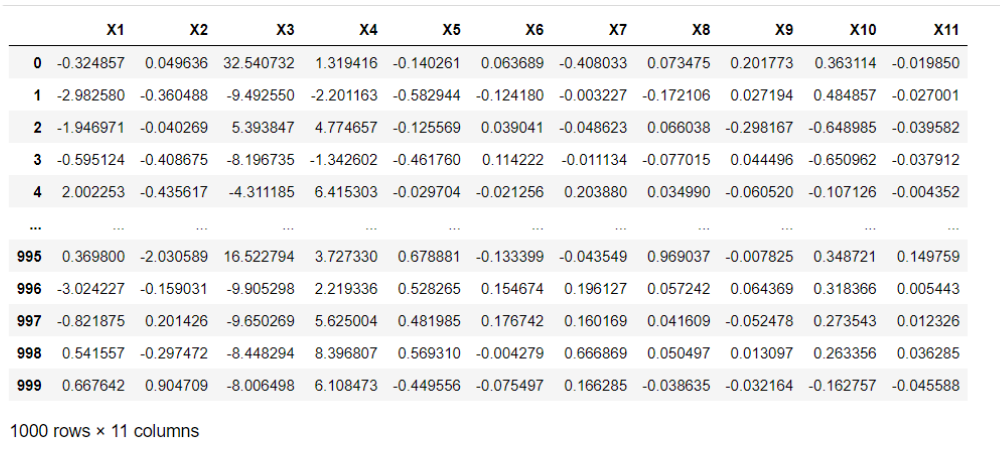
The neural network has 2 hidden layers with 'ReLU' activation and an output layer with 'sigmoid' activation.It is trained for 15 epochs and used a batch size of 32. After fitting the models, we applied explainers LIME and SHAP to explain the intances.For each explainer we got weightes for each instance which we collected in excel sheet.The below figures are just examples to show how exactly these weights look like:



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
0	-1.025471	0.000000	34.986173	1.815226	-0.547577	-0.273573	-0.295708	0.251558	0.655979	0.944197	-0.201051
1	-4.349684	-0.761111	-13.416312	-1.693988	-0.827905	1.122351	1.251025	-1.491602	0.976938	0.197671	0.000000
2	-4.252173	-0.171213	33.485739	8.779011	-0.342034	0.277446	-0.221775	0.180441	0.000000	-0.617428	-0.297551
3	-0.933642	-0.308613	-9.067880	-1.965365	-0.606826	0.000000	-0.935251	-0.803294	1.245337	-0.384566	-0.302551
4	0.930546	-0.333799	-8.411048	8.912571	-1.458130	-0.562967	0.681895	0.199073	0.000000	-0.853421	-0.304453
...	...	...	...	...	...	...	...	...	...	...	...
995	0.469912	-1.172370	34.243829	8.423341	0.526527	-0.109568	0.000000	-0.123147	0.237415	0.371446	0.499366
996	-5.436959	-0.033298	-13.942042	3.275685	0.000000	-0.245385	0.177238	0.601557	-0.973303	0.458538	0.263186
997	-0.339385	-0.435647	-14.282174	8.266489	0.502221	0.000000	0.577142	-0.996809	-0.228347	0.262979	0.230215
998	-0.738107	-0.375839	-13.288672	8.869813	0.200253	0.872638	0.602837	0.000000	1.252917	0.481227	0.307447
999	0.826548	-0.269947	-12.447930	8.868340	-0.501402	0.000000	0.466669	0.101319	-0.555687	-0.662921	-0.278766

1000 rows × 11 columns

Figure 7.1: LIME Weights for first Dataset simulated from equation 1



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
0	-0.324857	0.049636	32.540732	1.319416	-0.140261	0.063689	-0.408033	0.073475	0.201773	0.363114	-0.019850
1	-2.982580	-0.360488	-9.492550	-2.201163	-0.582944	-0.124180	-0.003227	-0.172106	0.027194	0.484857	-0.027001
2	-1.946971	-0.040269	5.393847	4.774657	-0.125569	0.039041	-0.048623	0.066038	-0.298167	-0.648985	-0.039582
3	-0.595124	-0.408675	-8.196735	-1.342602	-0.461760	0.114222	-0.011134	-0.077015	0.044496	-0.650962	-0.037912
4	2.002253	-0.435617	-4.311185	6.415303	-0.029704	-0.021256	0.203880	0.034990	-0.060520	-0.107126	-0.004352
...	...	...	...	...	...	...	...	...	...	...	...
995	0.369800	-2.030589	16.522794	3.727330	0.678881	-0.133399	-0.043549	0.969037	-0.007825	0.348721	0.149759
996	-3.024227	-0.159031	-9.905298	2.219336	0.528265	0.154674	0.196127	0.057242	0.064369	0.318366	0.005443
997	-0.821875	0.201426	-9.650269	5.625004	0.481985	0.176742	0.160169	0.041609	-0.052478	0.273543	0.012326
998	0.541557	-0.297472	-8.448294	8.396807	0.569310	-0.004279	0.666869	0.050497	0.013097	0.263356	0.036285
999	0.667642	0.904709	-8.006498	6.108473	-0.449556	-0.075497	0.166285	-0.038635	-0.032164	-0.162757	-0.045588

1000 rows × 11 columns

Figure 7.2: SHAP Weights for first Dataset simulated from equation 1

7.1 Evaluation

True Positives(TP) and True Negatives(TN)
In the context of the equations, there are 11 variables, some of which are actively utilized in calculating the response variable, while the remaining variables are redundant. So as we already know which are important and which are not, based on the weights from LIME and SHAP for each dataset we calculated TP and TN.

7.1.1 Methodology

Let us understand with the help of equation

$$y = 2x_1 + 3log(x_2) + 4x_3^2 - 8\sqrt{x_4} + cos(x_5) + 0.3x_6 + \epsilon \tag{7.1.1}$$

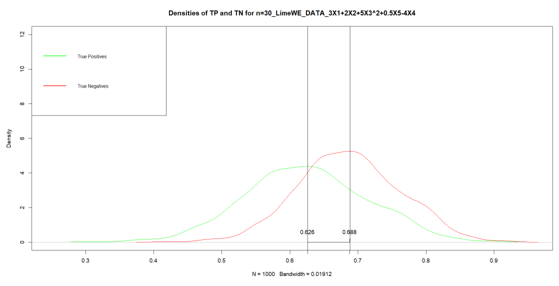
Here, only x_1,x_2,x_3,x_4,x_5,x_6 and ϵ were used to get response y , but while fitting model i.e Random forest regressor etc. It also took x_7,x_8,x_9,x_{11} into account. Most of the time we determine which features are redundant and assess the performance of the model, We calculate true positives and true negatives. But here we want to assess the performance of the explainers based on the weigths they provided for the features. Once we obtain the feature weights provided by the explainers (LIME and SHAP) for each feature. These weights indicate the importance or contribution of each feature towards the model's predictions. To assess the performance of the explainers (LIME and SHAP) in identifying important and redundant features, we ranked the feature weights provided by the explainers. This allowed us to create two subsets: one consisting of features deemed important and the other consisting of features deemed not important. Importantly, we already have prior knowledge about which features are truly important and which are redundant. Using this information, we proceeded to calculate true positives (TP) and true negatives (TN) to evaluate the performance of the explainers in correctly identifying important and redundant features. True positives (TP) were calculated as the number of features correctly identified as important by the explainers and in agreement with the ground truth. In other words, TP represents the instances where both the explainers and the ground truth correctly identified a feature as important. True negatives (TN) were calculated as the number of features correctly identified as not important by the explainers and in agreement with the ground truth. TN represents the instances where both the explainers and the ground truth correctly identified a feature as redundant. By calculating TP and TN, we were able to quantitatively evaluate the performance of the explainers in correctly identifying important and redundant features. It's important to note that TP and TN provide insights into the explainers' accuracy in identifying features. However, to obtain a more comprehensive assessment, additional metrics such as precision, recall, or F1-score can be calculated. These metrics consider false positives and false negatives, which further contribute to the evaluation of the explainers' performance. In conclusion, by comparing the rankings generated by the explainers with the ground truth knowledge about important and redundant features, we were able to calculate TP and TN. These metrics allowed us to assess the performance of the explainers in correctly identifying important and redundant features and provided valuable insights into their effectiveness in feature explanation.

In order to gain further insights into the performance of the explainers in identifying important and redundant features, we conducted two experiments. The first experiment aimed to investigate how changes in observation size affect the accuracy of the explainers in correctly identifying these features. The second experiment explored how varying the proportion of important features impacts the explainers' performance in the same task.

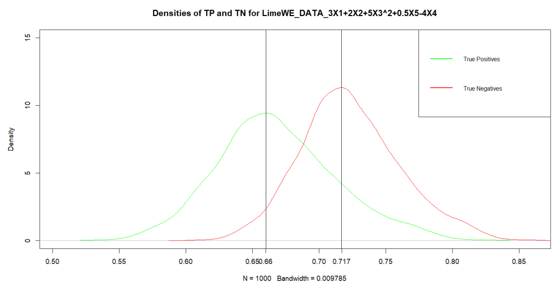
Experiment 1: Change in Sample Size
To examine the effect of observation size on the explainers' accuracy, we systematically varied the number of samples used for explanation. We started by using a sample size equal to 30 and gradually increased it to obtain a range of sample sizes.

For each sample size, we applied the explainers (LIME and SHAP) to rank the feature weights and identify important and redundant features. We then compared these rankings with the ground truth knowledge to calculate true positives (TP) and true negatives (TN).

By analyzing the relationship between sample size and the accuracy of the explainers, we gained insights into how changes in sample size impact the explainers' ability to correctly identify important and redundant features.

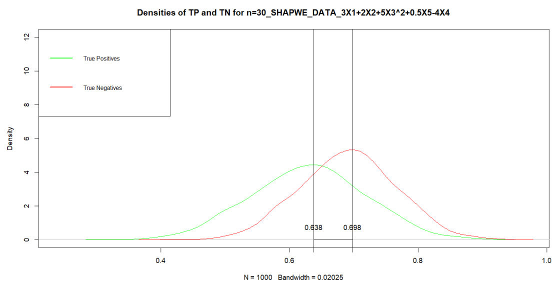


((a)) TP and TN for sample size=30

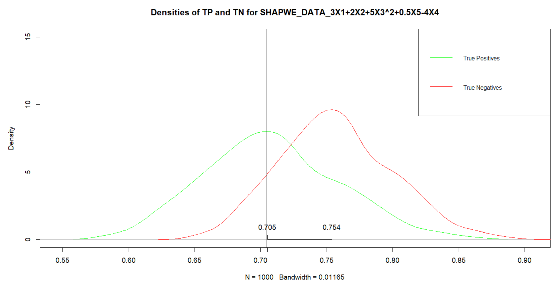


((b)) TP and TN for sample size=100

Figure 7.3: Graph of LIME WEIGHTS for Equation $Y = 3X_1 + 2X_2 + 5X_3^2 - 4X_4 + 0.5X_5$



((a)) TP and TN for sample size=30



((b)) TP and TN for sample size=100

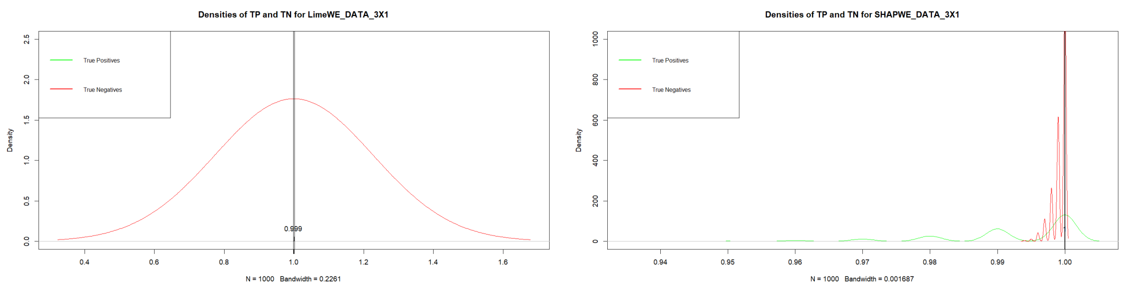
Figure 7.4: Graph of SHAP WEIGHTS for Equation $Y = 3X_1 + 2X_2 + 5X_3^2 - 4X_4 + 0.5X_5$

Experiment 2: Changing Proportion of Important Features
In the second experiment, we focused on understanding the impact of changing the proportion of important features on the explainers' performance.

Starting with a baseline proportion of important features, we systematically adjusted this proportion by modifying the dataset. We generated multiple datasets with varying proportions of important features, ensuring that the overall number of features remained constant.

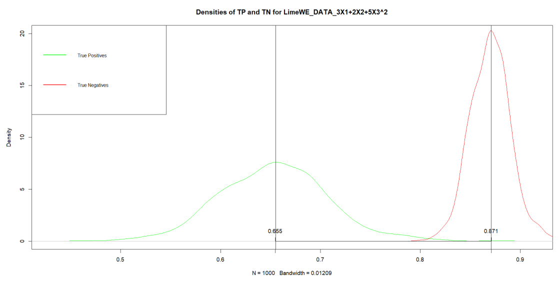
Similar to Experiment 1, we utilized the explainers to rank the feature weights and determine important and redundant features for each dataset. Comparing these rankings with the ground truth, we calculated true positives (TP) and true negatives (TN). Analyzing the relationship between the proportion of important features and the explainers' accuracy allowed us to assess how changes in feature importance distribution affect the explainers' performance in identifying important and redundant features. By conducting these experiments, we aimed to gain deeper insights into how sample size and the proportion of important features influence the accuracy of the explainers in correctly identifying important and redundant features. The results of these experiments will provide valuable information for optimizing the explainers' performance and understanding their limitations in different scenarios.

In conclusion, through the systematic manipulation of sample size and the proportion of important features, we evaluated the explainers' accuracy in identifying important and redundant features.

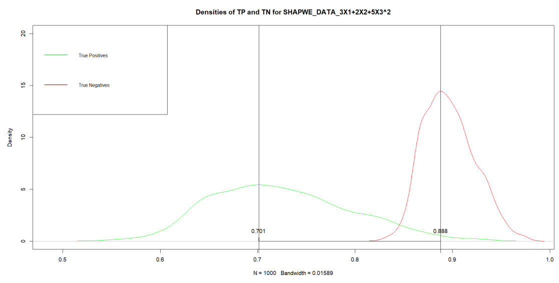


((a)) LIME WEIGHTS ((b)) SHAP WEIGHTS

Figure 7.5: Graph for Equation $Y = 3X_1$

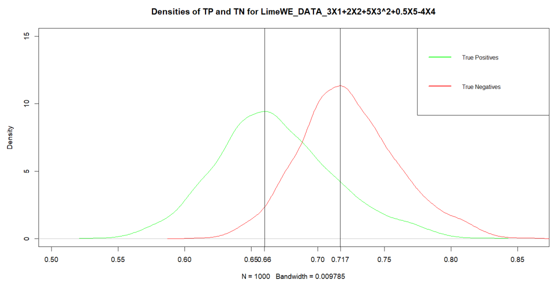


((a)) LIME WEIGHTS

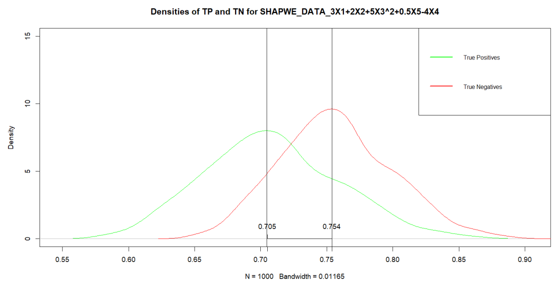


((b)) SHAP WEIGHTS

Figure 7.6: Graph for Equation $Y = 3X_1 + 2X_2 + 5X_3^2$

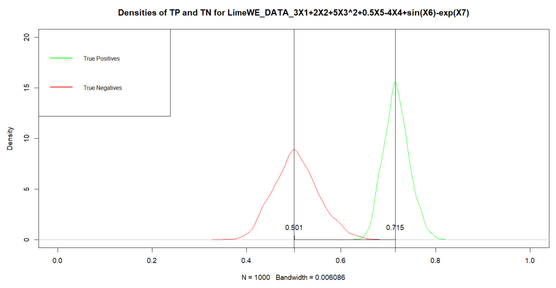


((a)) LIME WEIGHTS

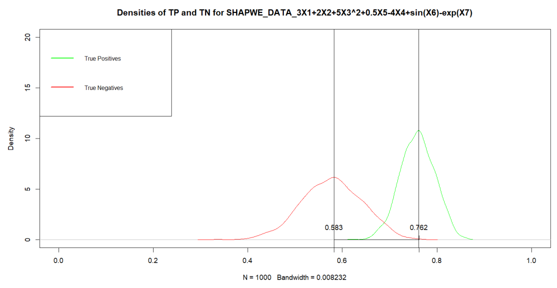


((b)) SHAP WEIGHTS

Figure 7.7: Graph for Equation $Y = 3X_1 + 2X_2 + 5X_3^2 - 4X_4 + 0.5X_5$

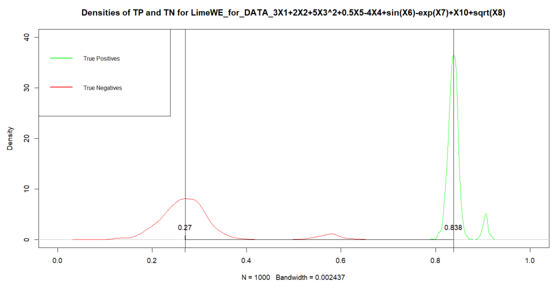


((a)) LIME WEIGHTS

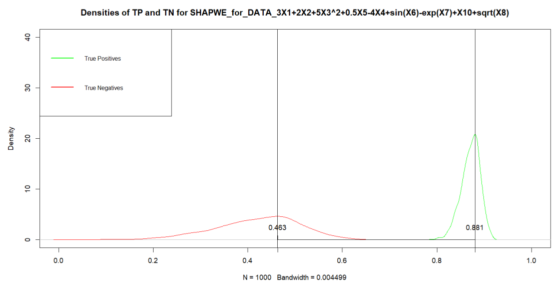


((b)) SHAP WEIGHTS

Figure 7.8: Graph for Equation $Y = 3X_1 + 2X_2 + 5X_3^2 - 4X_4 + 0.5X_5 + sin(X_6) - exp(X_7)$



((a)) LIME WEIGHTS



((b)) SHAP WEIGHTS

Figure 7.9: Graph for Equation $Y = 3X_1 + 2X_2 + 5X_3^2 - 4X_4 + 0.5X_5 + sin(X_6) - exp(X_7) + sqrt(X_8)$

7.2 Comparison Using Different Matrics

To measure the similarity between two vectors(e.g., between two sets of explanations or between an explanation and the true model weights), we use Pearson correlation, Spearman rank correlation and cosine distance. Pearson correlation and Spearman rank correlation ranges between [-1,1] and cosine distance ranges between [-1,1]. For all three metrics, the higher the value, the more similar two given vectors are.

The tables given below shows the results that we got for different matrics for the 06 datsets. First column gives Pearson's Correlation, second column gives Spearman's Rank Correlation and third column gives Cosine Distance. First table gives results for datasets simulated by first equation, Second table gives results for datasets simulated by second equation, Third table gives results for datasets simulated by third equation, Fourth table gives results for datasets simulated by fourth equation, Fifth table gives results for datasets simulated by fifth equation and Sixth table gives results for datasets simulated by sixth equation. These results are only for model XGBoost and observation size = 100.

Table 7.1: Equation 1

	Corr	Rank	Cosine
Mean	0.90566	0.67356	0.90510
Min	0.74109	0.45509	0.74394
Max	0.96983	0.88409	0.97109
Median	0.91264	0.67727	0.91215
25th %ile	0.89017	0.63639	0.88970
75th %ile	0.92848	0.71709	0.92797

Table 7.2: Equation 2

	Corr	Rank	Cosine
Mean	0.89125	0.71081	0.89050
Min	0.72204	0.49227	0.72368
Max	0.96087	0.85218	0.96103
Median	0.89617	0.71441	0.89518
25th %ile	0.87537	0.67539	0.87482
75th %ile	0.91425	0.74877	0.91375

■

Table 7.3: Equation 3

	Corr	Rank	Cosine
Mean	0.90501	0.36918	0.90518
Min	0.50703	0.18145	0.52498
Max	0.99376	0.60618	0.99370
Median	0.97278	0.35745	0.97238
25th %ile	0.85301	0.31763	0.85281
75th %ile	0.98208	0.41007	0.98177

Table 7.4: Equation 4

	Corr	Rank	Cosine
Mean	0.77332	0.49164	0.77265
Min	0.24729	0.20589	0.29110
Max	0.97265	0.80527	0.97150
Median	0.79077	0.49259	0.79001
25th %ile	0.71381	0.41339	0.71350
75th %ile	0.85918	0.56968	0.85788

Table 7.5: Equation 5

	Corr	Rank	Cosine
Mean	0.85201	0.78307	0.84014
Min	0.62584	0.565	0.61610
Max	0.93754	0.90232	0.93137
Median	0.85928	0.79140	0.84744
25th %ile	0.83247	0.75052	0.81907
75th %ile	0.88127	0.81998	0.86903

Table 7.6: Equation 6

	Corr	Rank	Cosine
Mean	0.90450	0.81210	0.89889
Min	0.79126	0.67927	0.78179
Max	0.95953	0.90572	0.95623
Median	0.90828	0.81733	0.90267
25th %ile	0.89259	0.78811	0.88627
75th %ile	0.92166	0.84172	0.91623

Chapter 8

CONCLUSIONS

This study reviewed explainability methods LIME and SHAP to help understand how machine learning models work. It explored different aspects of explainability, such as local explanations and feature importance. The study found that while two methods may rank features differently, they tend to agree on the most important ones.

The size of the sample used for analysis has limited influence on the consensus between LIME and SHAP regarding feature importance. Regardless of the sample size, both LIME and SHAP consistently provide similar rankings of feature importance. This suggests that the relative significance of features remains relatively stable and is not significantly affected by the number of instances in the dataset. Regardless of the sample size, these explainability methods consistently highlight the features that have the most significant impact on the model's predictions.

In our study, we evaluated the impact of three models: Random Forest, XGBoost, and Artificial Neural Network (ANN) which gave different model accuracies. The ANN model demonstrated superior accuracy compared to the other models. We observed that as the model accuracy increased, the similarity between the LIME weights and the SHAP weights also increased. Specifically, the ANN model resulted in more similar feature importance weights when compared to the other models.

These findings suggest that the choice of model influences the agreement between LIME and SHAP in determining feature importance. The higher accuracy achieved by the ANN model contributes to a closer alignment between the explanations provided by LIME and SHAP. It is important to consider the model's performance when interpreting and comparing the results obtained from different explainability methods.

The introduction of an interaction term in the dataset can impact the agreement between LIME and SHAP in terms of feature importance. Interaction terms represent the combined effect of two or more features, and they can influence the relative importance of individual features. Therefore, when interaction terms are included, LIME and SHAP may provide different rankings for feature importance, reflecting the influence of these interactions.

When averaging out the local explanations provided by LIME and SHAP to obtain global explanations for a single feature, there can be a drastic change in the correlations and cosine distance. Local explanations capture the importance of features for individual instances, while global explanations aim to provide an overall understanding of feature importance across the entire dataset. Averaging out local explanations can lead to a shift in the relative importance of features, resulting in changes in the correlation between LIME and SHAP weights.

In our investigation, we aimed to explore the influence of changing the proportion of important features on the performance of explainability methods. We established a baseline proportion of important features, initially set at 10%, and systematically adjusted this proportion by manipulating the dataset. Multiple datasets were generated, each with different proportions of important features, while maintaining a constant total number of features.

As the number of independent variables increased, we observed a decrease in both true positives (TP) and true negatives (TN). This suggests that the ability of the explainers to accurately identify important and non-important features diminishes as the complexity of the dataset increases.

Comparing the performance of the explainers, we found that SHAP exhibited slightly better accuracy than LIME overall. Although both methods provided valuable insights into feature importance.

These findings highlight the importance of considering the proportion of important features when applying explainability methods. As the proportion changes, the performance of the explainers may be affected, and selecting the most suitable method becomes crucial for accurate feature interpretation and understanding of the underlying model.

Chapter 9

LIMITATIONS AND FUTURE SCOPE

Feature Correlation:

LIME and SHAP, two popular methods for explaining machine learning models, assume that features are independent, which may not be true in real-world situations. When features are correlated, these methods may not accurately reflect the true importance of each feature in their explanations. This study focuses on uncorrelated features and one dataset with correlated datasets, but further can be explored by taking such datasets with correlated features and get explanations in such cases.

High-Dimensional Data:

LIME and SHAP can face challenges when dealing with high-dimensional data. The computational cost, time and interpretability of explanations may become less effective as the number of features increases.

Sensitivity to Perturbations:

LIME and SHAP explanations are based on **perturbing** input instances to measure feature importance. The choice of perturbation strategy can impact the explanations, and the explanations themselves may be sensitive to small changes in the perturbation process.

Lack of Ground Truth:
Evaluating the quality of explanations is challenging since there is no definitive "ground truth" for interpretability. Different evaluation metrics may have their own biases or limitations, making it difficult to objectively compare the performance of LIME and SHAP.

Human Interpretability:
Although LIME and SHAP provide explanations that are machine-interpretable, they may not always align with human intuition. The explanations might not fully capture the context, domain knowledge, or causal relationships that humans would expect.
Considering these limitations, it is important to carefully design the evaluation framework, select appropriate evaluation metrics, and acknowledge the specific constraints and assumptions of LIME and SHAP when comparing their performance in explainability in machine learning models.

While our analysis initially focused on LIME and SHAP, there are several other advanced explainability methods that can further enhance our understanding of feature importance and redundancy in the model. By incorporating these additional methods into analysis, we can gain a more comprehensive perspective on the underlying factors that contribute to the model's predictions. The following explainability methods can included to extended analysis, such as C-Lime, Kernal-SHAP, Vanilla gradient, Smooth Grad and Integrated gradient etc.

Bibliography

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145%2F2939672.2939785`.

Montgomery Flora, Corey Potvin, Amy McGovern, and Shawn Handler. Comparing explanation methods for traditional machine learning models part 1: An overview of current methods and quantifying their disagreement. *arXiv preprint arXiv:2211.08943*, 2022.

Patrick Hall, Wen Phan, and Sri Satish Ambati. Ideas on interpreting machine learning. *O'Reilly. Accessed: Mar*, 25:2019.

Tessa Han, Suraj Srinivas, and Himabindu Lakkaraju. Which explanation should i choose? a function approximation perspective to characterizing post hoc explanations. *arXiv preprint arXiv:2206.01254*, 2022.

Uday Kamath and John Liu. *Explainable Artificial Intelligence: An Introduction to Interpretable Machine Learning*. Springer, 2021.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

Appendix A

APPENDIX

Acivation functions

A1.1 Linear Function

Equation : Linear function has the equation similar to as of a straight line i.e. y = x No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
Range : -inf to +inf
Uses : Linear activation function is used at just one place i.e. output layer.
Issues : If we will differentiate linear function to bring non-linearity, result will no more depend on input "x" and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.
(`https://www.geeksforgeeks.org/activation-functions-neural-networks/`)

A1.2 ReLU (Rectified linear unit)

In the context of artificial neural networks, the rectifier or ReLU (rectified linear unit) activation function is an activation function defined as the positive part of its argument:

$$f(x) = x^+ = \max(0,x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \quad f'(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x < 0. \end{cases}$$

where x is the input to a neuron. Rectifying activation functions were used to separate specific excitation and unspecific inhibition in the neural abstraction pyramid, which was trained in a supervised way to learn several computer vision tasks.
(see `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)`)

A1.3 Sigmoid or Logistic Activation Function

- It is a function which is plotted as 'S' shaped graph.

- Equation : $s = \frac{1}{1+e^{-x}}$

- Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.

- Value Range : 0 to 1

- Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

(see `https://www.geeksforgeeks.org/activation-functions-neural-networks/`)

A1.4 Tanh Function

- The activation that works almost always better than sigmoid function is Tanh function also known as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

- Equation :- $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$.
- Value Range : -1 to +1
- Nature :- non-linear
- Uses :- Usually used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier. (see `https://www.geeksforgeeks.org/activation-functions-neural-networks/`)

Metrics

A2.1 Pearson's Correlation

It assesses how closely the data points of the two variables align on a straight line. The correlation coefficient ranges between -1 and +1, where -1 indicates a perfect negative linear relationship, +1 indicates a perfect positive linear relationship, and 0 indicates no linear relationship.
The formula for Pearson's correlation coefficient, r, is as follows:

$$r = \frac{\sum((X_i - \bar{X})(Y_i - \bar{Y}))}{\sqrt{\sum(X_i - \bar{X})^2}\sqrt{\sum(Y_i - \bar{Y})^2}}$$

In this formula, the variables are represented as follows:

X_i : Individual data points of the variable X

Y_i : Individual data points of the variable Y

$\bar{X}$: Mean (average) of variable X

$\bar{Y}$: Mean (average) of variable Y

A2.2 Spearman's Rank Correlation

It indicats how well the relationship can be represented using a monotonic function.
To calculate Spearman's rank correlation coefficient, ranks are assigned to each observation of the variables. Ranks are based on the value order, where the smallest value gets a rank of 1, the next smallest gets a rank of 2, and so on. In case of ties (when multiple observations share the same value), tied observations receive the average of the ranks they would have received.

The formula for Spearman's rank coefficient is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

ρ = Spearman's rank correlation coefficient

di = Difference between the two ranks of each observation

n = Number of observations

The Spearman Rank Correlation can take a value from +1 to -1 where,

A value of +1 means a perfect association of rank.
A value of 0 means that there is no association between ranks.
A value of -1 means a perfect negative association of rank.

A2.3 Cosine Distance

In data analysis, cosine similarity is a measure of similarity between two non-zero vectors defined in an inner product space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belongs to the interval $[-1, 1]$. The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

Given two n-dimensional vectors of attributes, **A** and **B**, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude formula:

cosine distance $= \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$

The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.
(see `https://en.wikipedia.org/wiki/Cosine_similarity`)

R Codes

A3.1 Data Generation

```
rm(list=ls())
set.seed(123)
n=1000
D=matrix(nrow=n,ncol=11)
View(D)
D[,1]=rnorm(n,0,1)
D[,2]=rgamma(n,1,2)
D[,3]=rexp(n,1)
D[,4]=rpois(n,2.5)
D[,5]=rnorm(n,1,2)
D[,6]=rgamma(n,3,4)
D[,7]=runif(n,0,1)
D[,8]=rexp(n,3)
D[,9]=rbinom(n,100,0.7)
D[,10]=rnorm(n,0,1)
D[,11]=ifelse(D[,5]<0,0,1)
Y1=3*D[,1]+2*D[,2]+5*(D[,3])^2+(0.5)*D[,5]+sin(D[,6])-4*D
   [,4]-exp(D[,7])+D[,10]
D1=cbind(D,Y1)
View(D6)
Y2=2*D[,1]+3*log(D[,2])+4*(D[,3])^2-8*sqrt(D[,4])+cos(D
   [,5])+0.3*D[,1]*D[,4]+D[,10]
D2=cbind(D,Y2)
Y3=D[,1]^2+(D[,2])+sqrt(D[,3])+log(D[,4])+exp(D[,5])+2*
   tan(D[,6])+D[,11]+D[,10]
D3=cbind(D,Y3)
Y4=4*D[,1]+8*(D[,2])^2+0.7*(D[,3])^3+5*(D[,4])^4+9*(D[,1]
   *D[,2])+10*D[,2]*D[,3]-27*D[,3]*D[,4]-2*D[,1]*D[,2]*D
   [,3]*D[,4]+D[,10]
```

```
D4=cbind(D,Y4)
Y=4*D[,1]+3*(D[,2])^3+45*(D[,3])^5+log(D[,4])+1/(D[,5])+D
   [,10]
Y5=ifelse(Y<0,0,1)
D5=cbind(D,Y5)
Z=D[,1]-5*(D[,2])-7*(D[,3])+3*(D[,4])-2*D[,1]*D[,4]+D
   [,5]^2
p=1/(1+exp(-Z))
Y6=c()
for(i in 1:n)
{
  Y6[i]=rbinom(1,1,p[i])
}
D6=cbind(D,Y6)
```

A3.2 For first equation

```
install.packages("openxlsx")
library(openxlsx)
rm(list=ls())
set.seed(123)

v<- list()
for (i in 1:1000) {
  f=list()
  n=1000
  D=matrix(nrow=n,ncol=11)
  D[,1]=rnorm(n,0,1)
  D[,2]=rgamma(n,1,2)
  D[,3]=rexp(n,1)
  D[,4]=rpois(n,2.5)
  D[,5]=rnorm(n,1,2)
  D[,6]=rgamma(n,3,4)
  D[,7]=runif(n,0,1)
  D[,8]=rexp(n,3)
  D[,9]=rbinom(n,100,0.7)
  D[,10]=rnorm(n,0,1)
  D[,11]=ifelse(D[,5]<0,0,1)
  Y1=3*D[,1]+2*D[,2]+5*(D[,3])^2+(0.5)*D[,5]+sin(D[,6])-4
     *D[,4]-exp(D[,7])+D[,10]
  D1=cbind(D,Y1)
  f[[i]]=D1
  sheet_name <- paste0("Sheet", i)
  v[[sheet_name]] <- f[[i]]
}

write.xlsx(v, file = 'D:\\MSc\\SEM IV\\Project\\Data\\
   Dataset1.xlsx')
```

A3.3 For TP and TN LimeWE for DATA 3X1

```
rm(list=ls())
library(readxl)
path_to_file <- "D:\\MSc\\SEM IV\\Project\\Data\\FEATURE
   BASED\\Outputs\\After 1st Column\\LimeWE_for_DATA_3X1.
   xlsx"

# Get the names of all sheets in the workbook
sheet_names <- excel_sheets(path_to_file)

process_sheet <- function(sheet_name) {

  data <- read_excel(path_to_file, sheet = sheet_name)
  V <- list()
  for(i in 1:nrow(data)) {
    V[[i]] <- sort(abs(data[i,]),decreasing = T)
  }

  EI <- list()
  ENI <- list()
  for(i in 1:nrow(data)) {
    EI[[i]] <- V[[i]][1]
    ENI[[i]] <- V[[i]][-1]
  }

  AI <- c(1)
  ANI <- c(2,3,4,5,6,7,8,10,9,11)

  countI <- countNI <- numeric(nrow(data))
  for(i in 1:nrow(data)) {
    countI[i] <- length(intersect(as.numeric(substring(
       names(EI[[i]]), 2)), AI))/1
    countNI[i] <- length(intersect(as.numeric(substring(
       names(ENI[[i]]), 2)), ANI))/10
  }

  data.frame(countI = mean(countI), countNI = mean(
     countNI))
}

results <- lapply(sheet_names, process_sheet)
results_df1 <- do.call(rbind, results)

max_density1 <- max(density(results_df1[,1])$y)
TPmax_x <- density(results_df1[,1])$x[which(density(
   results_df1[,1])$y == max_density1)]

max_density2 <- max(density(results_df1[,2])$y)
TNmax_x <- density(results_df1[,2])$x[which(density(
   results_df1[,2])$y == max_density2)]


plot(density(results_df1[,1]),col="green",main="Densities
    of TP and TN for LimeWE_DATA_3X1",ylim=c(min=0,max
   =2.5))
lines(density(results_df1[,2]),col="red")
legend("topleft",legend=c("True Positives","True
   Negatives"),col=c("green","red"),lty=1,lwd=2,cex=0.85)
abline(v=c(TPmax_x,TNmax_x),axis=c(TPmax_x,TNmax_x))
#axis(1,at=c(round(TPmax_x,3),round(TNmax_x,3)),labels=c(
   round(TPmax_x,3),round(TNmax_x,3)),las=2)
axis(1, at = c(round(TPmax_x, 3), round(TNmax_x, 3)),
   labels = FALSE, xpd = TRUE)
axis(3, at = c(round(TPmax_x, 3), round(TNmax_x, 3)),
   labels = c(round(TPmax_x, 3), round(TNmax_x, 3)), pos
   = 0.005)
```

A3.4 For TP and TN LimeWE for DATA 3X1+2X2+5X3^2+ $0.5X5 - 4X4$

```
rm(list=ls())
library(readxl)
path_to_file <- "D:\\MSc\\SEM IV\\Project\\Data\\FEATURE
   BASED\\Outputs\\After 1st Column\\LimeWE_for_DATA_3X1
   +2X2+5X3^2+0.5X5-4X4.xlsx"

# Get the names of all sheets in the workbook
sheet_names <- excel_sheets(path_to_file)

process_sheet <- function(sheet_name) {

  data <- read_excel(path_to_file, sheet = sheet_name)
  V <- list()
  for(i in 1:nrow(data)) {
    V[[i]] <- sort(abs(data[i,]),decreasing = T)
  }

  EI <- list()
  ENI <- list()
  for(i in 1:nrow(data)) {
    EI[[i]] <- V[[i]][1:5]
    ENI[[i]] <- V[[i]][-c(1:5)]
  }

  AI <- c(1,2,3,4,5)
  ANI <- c(6,7,8,10,9,11)
```

```r
  countI <- countNI <- numeric(nrow(data))
  for(i in 1:nrow(data)) {
    countI[i] <- length(intersect(as.numeric(substring(
       names(EI[[i]]), 2)), AI))/5
    countNI[i] <- length(intersect(as.numeric(substring(
       names(ENI[[i]]), 2)), ANI))/6
  }

  data.frame(countI = mean(countI), countNI = mean(
     countNI))
}

results <- lapply(sheet_names, process_sheet)
results_df1 <- do.call(rbind, results)

max_density1 <- max(density(results_df1[,1])$y)
TPmax_x <- density(results_df1[,1])$x[which(density(
   results_df1[,1])$y == max_density1)]

max_density2 <- max(density(results_df1[,2])$y)
TNmax_x <- density(results_df1[,2])$x[which(density(
   results_df1[,2])$y == max_density2)]


plot(density(results_df1[,1]),col="green",main="Densities
    of TP and TN for LimeWE_DATA_3X1+2X2+5X3^2+0.5X5-4X4"
   ,ylim=c(min=0,max=2.5))
lines(density(results_df1[,2]),col="red")
legend("topleft",legend=c("True Positives","True
   Negatives"),col=c("green","red"),lty=1,lwd=2,cex=0.85)
abline(v=c(TPmax_x,TNmax_x),axis=c(TPmax_x,TNmax_x))
#axis(1,at=c(round(TPmax_x,3),round(TNmax_x,3)),labels=c(
   round(TPmax_x,3),round(TNmax_x,3)),las=2)
axis(1, at = c(round(TPmax_x, 3), round(TNmax_x, 3)),
   labels = FALSE, xpd = TRUE)
axis(3, at = c(round(TPmax_x, 3), round(TNmax_x, 3)),
   labels = c(round(TPmax_x, 3), round(TNmax_x, 3)), pos
   = 0.005)
```

Python Code

A4.1 Random Forest

LIME Weights for Dataset 1

```python
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import lime
from lime import lime_tabular
import re

import warnings
warnings.filterwarnings('ignore')

##Matrix for weight
Ls=[]
for m in range(1000):
    Ls.append("Lime_Weights"+str(m))

for n in range(1000):
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)
    DS.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6','X7',
       'X8', 'X9','X10', 'X11','Y'], axis='columns',
       inplace=True)
    matrix=np.zeros((100,11),dtype=float)
    lime_wets1=pd.DataFrame(matrix)
    lime_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
       ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
       inplace=True)
    Ls[n]=pd.DataFrame()

    ###Fittig model
    #Train-Test split
    X = DS.drop('Y', axis=1)
    y = DS['Y']
    np.random.seed(123)
    X_train, X_test, y_train, y_test = train_test_split(X
       , y, test_size=0.2, random_state=42)

    #Fitting Random Forest
    model = RandomForestRegressor(n_estimators = 10,
       random_state=42)
    model.fit(X_train, y_train)

    ###LIME Explainer
    explainer = lime_tabular.LimeTabularExplainer(
       training_data=np.array(X), categorical_features= "
       X11", categorical_names={"X11":['0','1']}, feature
       _names=X.columns,mode='regression')

    for i in range(100):
        exp = explainer.explain_instance(data_row=X.iloc[
           i], predict_fn=model.predict)
        weight=exp.as_list()
        wet=[(re.findall(r'X\d+',a)[0],b) for a,b in
           weight]
        wet.sort()
        Wet_Sorted_Final=sorted(wet,key=lambda x: int(x
           [0][1:]))

        for a,b in Wet_Sorted_Final:
            if a=='X1':
                lime_wets1['X1'][i]=b
            elif a=='X2':
                lime_wets1['X2'][i]=b
            elif a=='X3':
                lime_wets1['X3'][i]=b
            elif a=='X4':
                lime_wets1['X4'][i]=b
            elif a=='X5':
                lime_wets1['X5'][i]=b
            elif a=='X6':
                lime_wets1['X6'][i]=b
            elif a=='X7':
                lime_wets1['X7'][i]=b
            elif a=='X8':
                lime_wets1['X8'][i]=b
            elif a=='X9':
                lime_wets1['X9'][i]=b
            elif a=='X10':
                lime_wets1['X10'][i]=b
            elif a=='X11':
                lime_wets1['X11'][i]=b

    Ls[n]=lime_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\Lime_WeightsG1.xlsx'
with pd.ExcelWriter(file_name) as writer:  # doctest: +
   SKIP
    for m in range(1000):
        Ls[m].to_excel(writer, sheet_name='Sheet_'+str(m)
           )
```

SHAP Weights for Dataset 1

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import shap

Ls=[]
```

```
for m in range(1000):
    Ls.append("Shap_Weights"+str(m))

for n in range(1000):
    ##Matrix for weight
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)
    DS.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6','X7',
       'X8', 'X9','X10', 'X11','Y'], axis='columns',
       inplace=True)

    ###Fittig model
    #Train-Test split
    np.random.seed(123)
    X = DS.drop('Y', axis=1)
    y = DS['Y']
    X_train, X_test, y_train, y_test = train_test_split(X
       , y, test_size=0.2, random_state=42)

    #Fitting Random Forest
    model = RandomForestRegressor(n_estimators = 10,
       random_state=42)
    model.fit(X_train, y_train)

    ###SHAP Explainer
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values( X)

    #Storing weights for each oservation in matrix
    matrix=np.zeros((100,11),dtype=float)
    Shap_wets1=pd.DataFrame(matrix)
    Shap_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
       ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
       inplace=True)
    for i in range(100):
        shap_wet=shap_values[i,:]
        for j in range(11):
            Shap_wets1.iat[i,j]=(shap_wet[j])

    Ls[n]=Shap_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\SHAP_WeightsG1.xlsx'
with pd.ExcelWriter(file_name) as writer:  # doctest: +
   SKIP
    for m in range(1000):
        Ls[m].to_excel(writer, sheet_name='Sheet'+str(m))
```

A4.2 XGBoost

LIME Weights for Dataset 1

```
import numpy as np
import pandas as pd
import xgboost as xg
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')

import lime
from lime import lime_tabular
import re

##Matrix for weight
Ls=[]
for m in range(1000):
    Ls.append("Lime_Weights"+str(m))

for n in range(1000):
    print(n)
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)
    DS.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6','X7',
       'X8', 'X9','X10', 'X11','Y'], axis='columns',
       inplace=True)

    matrix=np.zeros((100,11),dtype=float)
    lime_wets1=pd.DataFrame(matrix)
    lime_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
       ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
       inplace=True)
    Ls[n]=pd.DataFrame()

    ###Fittig model
    #Train-Test split
    X = DS.drop('Y', axis=1)
    y = DS['Y']
    np.random.seed(123)
    X_train, X_test, y_train, y_test = train_test_split(X
       , y, test_size=0.2, random_state=42)
    #Fitting XGBoost
    xgb_r = xg.XGBRegressor(objective ='reg:linear', max_
       depth=2,
                  n_estimators = 100,learning_rate=0.4,
                     seed = 123)
    xgb_r.fit(X_train, y_train)
    ###LIME Explainer
    explainer = lime_tabular.LimeTabularExplainer(
       training_data=np.array(X), categorical_features= "
       X11", categorical_names={"X11":['0','1']},feature_
       names=X.columns,mode='regression')

    for i in range(100):
        exp = explainer.explain_instance(data_row=X.iloc[
           i], predict_fn=xgb_r.predict)
        weight=exp.as_list()
        wet=[(re.findall(r'X\d+',a)[0],b) for a,b in
           weight]
        wet.sort()
        Wet_Sorted_Final=sorted(wet,key=lambda x: int(x
           [0][1:]))

        for a,b in Wet_Sorted_Final:
            if a=='X1':
                lime_wets1['X1'][i]=b
            elif a=='X2':
                lime_wets1['X2'][i]=b
            elif a=='X3':
                lime_wets1['X3'][i]=b
            elif a=='X4':
                lime_wets1['X4'][i]=b
            elif a=='X5':
                lime_wets1['X5'][i]=b
            elif a=='X6':
                lime_wets1['X6'][i]=b
            elif a=='X7':
                lime_wets1['X7'][i]=b
            elif a=='X8':
                lime_wets1['X8'][i]=b
            elif a=='X9':
                lime_wets1['X9'][i]=b
            elif a=='X10':
                lime_wets1['X10'][i]=b
            elif a=='X11':
                lime_wets1['X11'][i]=b

    Ls[n]=lime_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\XGBoost\\XGLime_WeightsG1.xlsx'
with pd.ExcelWriter(file_name) as writer:  # doctest: +
   SKIP
    for m in range(1000):
        Ls[m].to_excel(writer, sheet_name='Sheet_'+str(m)
           )
```

SHAP Weights for Dataset 1

```
import numpy as np
import pandas as pd
import xgboost as xg
from sklearn.model_selection import train_test_split
import shap

import warnings
warnings.filterwarnings('ignore')

Ls=[]
for m in range(1000):
    Ls.append("Shap_Weights"+str(m))

for n in range(1000):
    ##Matrix for weight
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)
    DS.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6','X7',
       'X8', 'X9','X10', 'X11','Y'], axis='columns',
       inplace=True)

    ###Fittig model
    #Train-Test split
    X = DS.drop('Y', axis=1)
    y = DS['Y']
    np.random.seed(123)
    X_train, X_test, y_train, y_test = train_test_split(X
       , y, test_size=0.2, random_state=42)
    #Fitting XGBoost
    xgb_r = xg.XGBRegressor(objective ='reg:linear', max_
       depth=2,
                  n_estimators = 100,learning_rate=0.4,
                     seed = 123)
    xgb_r.fit(X_train, y_train)

    ###SHAP Explainer
    explainer = shap.TreeExplainer(xgb_r)
    shap_values = explainer.shap_values( X)

    #Storing weights for each oservation in matrix
    matrix=np.zeros((100,11),dtype=float)
    Shap_wets1=pd.DataFrame(matrix)
    Shap_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
       ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
       inplace=True)

    for i in range(100):
        shap_wet=shap_values[i,:]
        for j in range(11):
            Shap_wets1.iat[i,j]=(shap_wet[j])

    Ls[n]=Shap_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Datasets\\XGShap_WeightsG1.xlsx'
with pd.ExcelWriter(file_name) as writer:  # doctest: +
   SKIP
    for m in range(1000):
        Ls[m].to_excel(writer, sheet_name='Sheet'+str(m))
```

A4.3 ANN

LIME Weights for Dataset 1

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

import lime
from lime import lime_tabular

import re
import warnings
warnings.filterwarnings('ignore')

##Matrix for weight
Ls=[]
for m in range(1000):
    Ls.append("Lime_Weights"+str(m))

for n in range(1000):
    print(n)
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)                              DS.set_axis(['X1', '
       X2', 'X3','X4', 'X5', 'X6','X7', 'X8', 'X9','X10',
        'X11','Y'], axis='columns', inplace=True)

    matrix=np.zeros((100,11),dtype=float)
    lime_wets1=pd.DataFrame(matrix)
```

```
lime_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
   ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
   inplace=True)
Ls[n]=pd.DataFrame()
###Fittig model

#Train-Test split
X = DS.drop('Y', axis=1)
y = DS['Y']
np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(X
   , y, test_size=0.2, random_state=42)

###Fitting ANN
#Initializing the ANN
ann = Sequential()
#Adding First Hidden Layermm
ann.add(Dense(units=15, activation="tanh"))
# Adding Second Hidden Layer
ann.add(Dense(units=30, activation="tanh"))
# Adding Output Layer
ann.add(Dense(units=1,activation="linear"))
ann.compile(optimizer="adam",loss="mean_squared_error
   ")
ann.fit(x=X_train, y=y_train, epochs=15, batch_size
   =32,validation_data=(X_test,y_test),callbacks=
   EarlyStopping(monitor='val_loss',patience=4))

###LIME Explainer
explainer = lime_tabular.LimeTabularExplainer(
   training_data=np.array(X), categorical_features= "
   X11", categorical_names={"X11":['0','1']}, feature
   _names=X.columns,mode='regression')

for i in range(100):
    exp = explainer.explain_instance(data_row=X.iloc[
       i], predict_fn=ann.predict)
    weight=exp.as_list()
    wet=[(re.findall(r'X\d+',a)[0],b) for a,b in
       weight]
    wet.sort()
    Wet_Sorted_Final=sorted(wet,key=lambda x: int(x
       [0][1:]))

    for a,b in Wet_Sorted_Final:
        if a=='X1':
            lime_wets1['X1'][i]=b
        elif a=='X2':
            lime_wets1['X2'][i]=b
        elif a=='X3':
                lime_wets1['X3'][i]=b
            elif a=='X4':
                lime_wets1['X4'][i]=b
            elif a=='X5':
                lime_wets1['X5'][i]=b
            elif a=='X6':
                lime_wets1['X6'][i]=b
            elif a=='X7':
                lime_wets1['X7'][i]=b
            elif a=='X8':
                lime_wets1['X8'][i]=b
            elif a=='X9':
                lime_wets1['X9'][i]=b
            elif a=='X10':
                lime_wets1['X10'][i]=b
            elif a=='X11':
                lime_wets1['X11'][i]=b

    Ls[n]=lime_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\ANN\\ANNLime_WeightsG1.xlsx'
with pd.ExcelWriter(file_name) as writer:
    for m in range(1000):
        Ls[m].to_excel(writer, sheet_name='Sheet_'+str(m)
           )
```

SHAP Weights for Dataset 1

```
import pandas as pd
import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
import shap
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')

Ls=[]
for m in range(1):
    Ls.append("Shap_Weights"+str(m))

for n in range(1):
    ##Matrix for weight
    DS = pd.read_excel("C:\\Users\\Student\\Documents\\R
       Studio\\Datasets\\ReducedDatasetEq1.xlsx", sheet_
       name=n)                           #Importing ith sheet
```

```
        from excel workbook
    DS.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6','X7',
        'X8', 'X9','X10', 'X11','Y'], axis='columns',
        inplace=True)

    ###Fittig model
    #Train-Test split
    X = DS.drop('Y', axis=1)
    y = DS['Y']
    np.random.seed(123)
    X_train, X_test, y_train, y_test = train_test_split(X
        , y, test_size=0.2, random_state=42)
    #Fitting ANN
    # Initializing the ANN
    ann = Sequential()
    #Adding First Hidden Layermm
    ann.add(Dense(units=15, activation="tanh"))
    #Adding Second Hidden Layer
    ann.add(Dense(units=30, activation="tanh"))
    #Adding Output Layer
    ann.add(Dense(units=1,activation="linear"))
    ann.compile(optimizer="adam",loss="mean_squared_error
        ")
    ann.fit(x=X_train, y=y_train, epochs=15, batch_size
        =32,validation_data=(X_test,y_test),callbacks=
        EarlyStopping(monitor='val_loss',patience=4))

    ###SHAP Explainer
    explainer = shap.DeepExplainer(ann, X)
    shap_values = explainer.shap_values(X)

    #Storing weights for each oservation in matrix
    matrix=np.zeros((100,11),dtype=float)
    Shap_wets1=pd.DataFrame(matrix)
    Shap_wets1.set_axis(['X1', 'X2', 'X3','X4', 'X5', 'X6
        ','X7', 'X8', 'X9','X10', 'X11'], axis='columns',
        inplace=True)

   for i in range(100):
        shap_wet=shap_values[i,:]
        for j in range(11):
            Shap_wets1.iat[i,j]=(shap_wet[j])

    Ls[n]=Shap_wets1
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\ANN\\ANNShap_WeightsG123.xlsx'
with pd.ExcelWriter(file_name) as writer:
    for m in range(1):
        Ls[m].to_excel(writer, sheet_name='Sheet'+str(m))
```

A4.4 Pearson Correlation, Rank Correlation & Cosine Distance

```
from scipy.stats import spearmanr
from sklearn.metrics.pairwise import cosine_distances

Corr_SL=[]
Rank_SL=[]
Cosine_SL=[]
for i in range(1000):
    print(i)
    LIME = pd.read_excel("C:\\Users\\Student\\Documents\\
       R Studio\\Outputs\\Random Forest\\Lime_WeightsG2.
       xlsx", sheet_name=i)
    LIME.drop('Unnamed: 0', axis=1, inplace=True)
    SHAP = pd.read_excel("C:\\Users\\Student\\Documents\\
       R Studio\\Outputs\\Random Forest\\SHAP_WeightsG2.
       xlsx", sheet_name=i)
    SHAP.drop('Unnamed: 0', axis=1, inplace=True)

    Corr=LIME.corrwith(SHAP, axis = 1)
    Corr_SL.append(np.mean(Corr))

    # Take the absolute value of the dataframes
    LIME_abs = LIME.abs()
    SHAP_abs = SHAP.abs()

    # Calculate the rank correlation using Spearman's
       correlation coefficient
    rank_corr = LIME_abs.corrwith(SHAP_abs,axis = 1,
       method='spearman')
    Rank_SL.append(np.mean(rank_corr))

   # Calculate the cosine distance between the two
      vectors
    cosine_dist_matrix = cosine_distances(LIME, SHAP)

    # Print the cosine distance matrix
    cosine_dist=[]
    for a in range(100):
        cosine_dist.append(cosine_dist_matrix[a][a])

    Cosine_SL.append(1-np.mean(cosine_dist))

Final=pd.DataFrame()
Final['Corr_SL']=Corr_SL
Final['Rank_SL']=Rank_SL
Final['Cosine_SL']=Cosine_SL
file_name = 'C:\\Users\\Student\\Documents\\R Studio\\
   Outputs\\Random Forest\\Final2.xlsx'
```

```
with pd.ExcelWriter(file_name) as writer:
        Final.to_excel(writer, sheet_name='SF_Final')
```