FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Feature Importance for Black-box Models
Student:	Ard Kelmendi
Supervisor:	Ing. Veronika Maurerová
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2021/22

Instructions

Interpretability is getting more and more critical for Machine Learning Modeling. Especially in the healthcare and financial industry. For example, it is not good if the model predicts the patient will die, and the data scientist or doctor can’t explain how the model has made the decision.
It is easy to calculate feature importance for linear or logistic regression or tree algorithms - the feature importance is a side effect of the calculation of inner parameter values. However, there are plenty of models where feature importance can’t be easily calculated during training. Or the models are black-box, so a user can only use the model for training and scoring and nothing else.
The goal is:

1. Select and describe a feature importance calculation for black-box models.
2. Implement the algorithm to the H2O-3 Open Source Machine Learning platform.
3. Test functionality and performance.
4. Compare results with other open-source implementation.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague October 23, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Feature Importance for black-box models

Ard Kelmendi

Department of Theoretical Computer Science
Supervisor: Ing. Veronika Maurerova

January 7, 2021

Acknowledgements

This thesis would not have been possible without the inspiration and support of a number of wonderful individuals. I owe my deepest gratitude to my supervisor Ing. Veronika Maurerova for giving me this opportunity. Without her encouragement, and continuous guidance this thesis would hardly have been completed.

My deep and sincere gratitude to my family for their continuous and unparalleled love, help and support. I am forever indebted to my parents for their selfless encouragement to explore the directions in life that interest me and choosing my own destiny. I am grateful for my sisters for being there for me without hesitation. I dedicate this milestone to them.

Declaration

In Prague on January 7, 2021 . . . .

Citation of this thesis

Kelmendi, Ard. *Feature Importance for black-box models.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Důležitost proměnných je technika, která přiřazuje skóre vstupním proměnným (sloupcům strukturovaných dat) na základě jejich vlivu na predikování cílové proměnné. Sloupce datasetu, které jsou použity jako vstup do algoritmu strojového učení se nazývají proměnné. Některé vtupní proměnné můžou být více důležité než ostatní tím, že více ovlivňují cílovou proměnou. Globální senzitivní analýza přiřazuje hodnoty jednotlivým vstupním proměnným na základě jejich interakcích pri predikci s ohledem na cílovou proměnnou a poskytuje tak skóre pro interpretaci modelů. Cílem této bakalářské práce je popsat metodu Permutační důležitosti proměnných a implementovat tuto metodu do H2O-3 open-source Machine Learning platformy.

Klíčová slova Strojové učení, interpretovatelnost, black-box modely, Globální Senzitivní analýza, Permutační důležitost proměnných, One At a Time

Abstract

Feature importance is a technique that assigns a score to input features (tabular data columns) based on the influence of predicting the target feature. The columns of a dataset that servers as an input of the Machine Learning algorithms are called features. Some features may be more important than others giving more influence towards the output. Global Sensitivity Analysis quantifies the importance of model features and their interactions with respect to model output. Assigning different values to the features one at a time provides the user with a mapping score of importance to features to interpret the model. The aim of this bachelor's thesis is to describe Permutation Feature Importance and implement this method to the H2O-3 open-source Machine Learning platform.

Keywords Machine Learning, Interpretability, black-box Models, Global Sensitivity Analysis, Permutation Feature Importance, One At a Time

Contents

List of Figures

List of Tables

Introduction

Memory is a curious thing. The way the brain retrieves information is highly intriguing. One can no doubt recite the Alphabet from A to Z, how about spelling it backward? Shouldn't be a hard task since the brain already has the information, however, it is quite a challenging task. The brain stores and organizes information according to patterns, and throughout history humans have become able to see patterns, understand them, and even predict future events based on what is known at the present. Humans have become so efficient with finding patterns in nature that they have created machines and algorithms to do it for them, faster and with promising results.

The term Machine Learning (ML) refers to the automated detection of meaningful patterns in data [1]. ML assists computer systems in progressively improving their performance. ML algorithms automatically build a mathematical model using sample data to devise decisions without specifically being programmed make such decisions.

Understanding and trusting these mathematical models and their results lead to good science. Today's forces of innovation and competition are leading analysts and ML engineers to try ever-more-complex predictive modeling. Using such complex algorithms tend to lead to more accurate predictions, at the same time leading to difficulties in understanding why those predictions are made. Higher Accuracy almost always trades off the interpretation of the model, known as "black-box" models. The columns of the dataset will be referred as features or variables based on context.

This thesis aims to analyze the mathematical model to obtain which features of the input data have the highest influence on the construction of the ML model on predicting the target feature. Changing the values of the features one at a time gives insight into the relationship of the data. Upon using "black-box" algorithms that tend to lose interpretation of the model's prediction, Permutation Feature Importance (PFI) sets forth a different approach of understanding without having to retrain the model. It's implementation is within H2O-3 open-source Machine Learning platform.

Chapter 1

Machine Learning

The term Machine Learning first came up in 1952 from Arthur Samuel, who created a computer program that with a small amount of limited memory was able to design a scoring function attempting to measure the chances of each side winning, his program became harder to beat [2].

1.1 Brief history

ML is based on a model of brain interaction created in 1949 by Donald Hebb in his book titled The Organization of Behaviour. The book consists of Hebb's theories on neuron communication and excitement. Hebb mentioned "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the some of the second cell" [3]. Hebb's concepts can be translated into Artificial Neural Networks (NN).

Combining Donald Hebb's model of brain cell interaction with Arthur Samuel's ML contributions, Frank Rosenblatt in 1957 was able to build software to be later installed in a custom-build machine, called Mark 1 perception [4], constructed for image recognition. The mentioned machine was unable to detect many kinds of visual patterns.

With more advancement in both technology and investment on ML within the next decades, ML has come to be used daily for simple and complex problems. Stanford University defined ML as "the science of getting computers to act without being explicitly programmed" [2]. Modern ML models can be used to make predictions ranging from add selection, self-driving vehicles, outbreaks of disease, to rise and fall of stocks. ML today is responsible for some of the most significant advancements in technology.

1.2 Introduction to Machine Learning

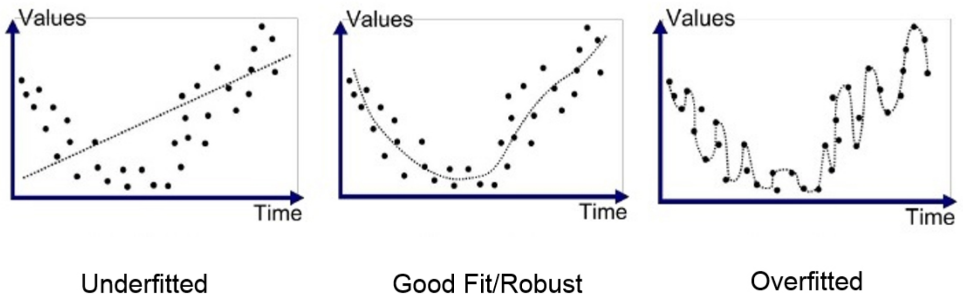
ML is a scientific study of algorithms and statistical models that computer systems detect (learn) meaningful patterns in data. In 1959 Arthur Samuel defined ML as a "Field of study that gives computers the ability to learn without being explicitly programmed" [5]. The model holds the learning behind the conclusion, it can be a mathematical representation of the relationship of data. The goal is to program a computer so that it can learn from sample data (training data see section 1.3). The input of ML is training data to be used from the program as experience. The output of ML can range from a prediction of some event to another computer program that can perform a task. In the past couple of decades when it comes to extracting data from large datasets ML has become a very common tool, seen as a positive factor on hectic data-driven decisions.

For ML theory, distinguishing learning mechanisms that result in superstition from useful learning is a central goal [6]. The machine, unlike humans, cannot rely on common sense to filter out random meaningless learning conclusions. One must provide well defined crisp principles that ensure useful conclusions. Incorporation of prior knowledge, biasing the learning process, is inevitable for the success of learning algorithms. Developing tools to express knowledge and understand the essential aspects of a specific field of inquiry, guiding the process, and evaluating the end product within the context of value and validity, translating it into a learning bias (set of assumptions that the learner uses to predict the output given datasets that it has not encountered), and quantifying the effect of such a bias on the success of the learning is a central theme of the theory of Machine Learning. The stronger the prior knowledge at the start of the learning process, the easier it becomes to learn from upcoming examples. On the other hand, the stronger these assumptions (prior knowledge), the less flexible the learning is, it is bound by these assumptions [6].

1.3 Train and Learn data

Given some data, called the training dataset, a model is trained. The model is built so that it will try to answer the question put based on the data provided, for instance, predict one (dependent) variable based on all the other (independent) variables or upon having data separated into clusters, predict which cluster the new data refers to and so on. After the model outputs a prediction a second dataset is needed to verify the performance of the model, the testing dataset. The model learned from the training data is used on the testing data discerning how well it predicts the variable in question. A robust model performs very similarly to the test dataset, whereas when a model predicts correctly on the training set and poorly on the testing set then the model is overfitting the data. In other words, the model does not capture the dominant trend of the data, not being able to predict a likely output for an input that it has never seen before. Underfitting is the case where the model has "not learned enough" from the training data, resulting in low generalization and unreliable predictions [7]. See Figure 1.1.

Figure 1.1: Underfit, robust, overfit [7].



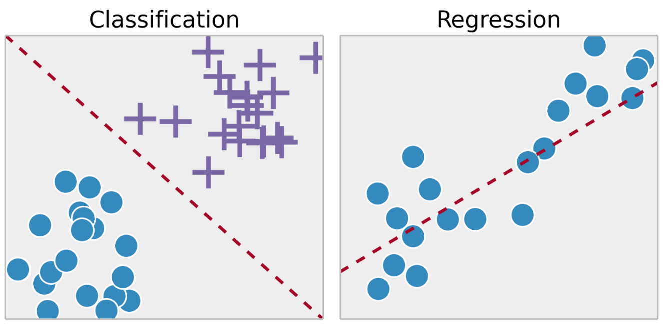
The loss function is a method of evaluating how well the algorithm models the given data (if the prediction deviates too much from the actual results, the loss function would consist of a large number). "A measurement of the cost to the performance task (and/or benefit) of making a prediction Y" [8].

1.4 Supervised and Unsupervised Learning

In ML there are two main categories of tasks: supervised and unsupervised. Supervised learning, in contrast to unsupervised, has prior knowledge of the output values; one has input variables and output variables and uses an algorithm to learn the mapping function from the input to the output. The goal of supervised learning is to learn a function, which uses sample data and knows desired outputs, that describes the best approximation of the relationship between sample data and desired outputs. On the other hand, Unsupervised learning which has no labeled outputs goal is to deduce and conclude with reasoning from data points a natural structure on data [9].

Supervised learning solves problems for classification or regression. When one wants to map an input to output labels, it refers to classification problems, whereas when one wants to map an input to a continuous output, it refers to regression problems. Even though different algorithms are used in both classification and regression, the goal for both is finding a specific interconnection of the input data in such a way that allows one to effectively produce correct predictions. While it stands true that one has more than one dataset, the model's output is determined using the training dataset only. The main considerations of supervised learning are model complexity and bias-variance trade-off. Model complexity indicates the complexity of the function attempting to be learned, in the case of linear regression it would be the degree of a polynomial. The complexity of the model is generally determined by the input data itself (one scenario that could lead to a higher complexity is a huge amount of data with many variables). High-complexity models with small-data tend to over-fit the data. Bias-variance trade-off is also linked with the generalization of a model, bias meaning the constant error term, and variance meaning the quantity the error can change in different training sets. Increasing bias leads to lower variance and vice versa.

Figure 1.2: Classification vs regression tasks [9].



1.4.1 Linear regression

There are many supervised learning algorithms. One being linear regression which solves regression tasks. It is a simple algorithm to demonstrate how to create a model that is easy to understand and quite reliable. In linear regression one wants to model the relationship between scalar response variables (dependent variable) y_i and a scalar or multivariate explanatory variable (independent variable) x_i. There exists a noise term e that affects the causality between the dependent variable and independent variable, $y_i = f(x_i) + e_i$.

$f(x_i)$ is some linear function of x_i, whereas e_i is a random variable. The goal of linear regression is to predict future values, predict values on missing existing data points (if any), and explain the change of the dependent variable based on the independent variables.

Assume the existence of p regressors $x_{i1}, \dots, x_{ip}$ determining each y_i for $i = 1, \dots, n$ and adopt a linear regression model in the form of a general line,

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + e_i \\ &= \begin{bmatrix} x_{i1}, \dots, x_{i_p} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ \beta_p \end{bmatrix} + e_i \\ &= \boldsymbol{x_i^T} \boldsymbol{\beta} + \boldsymbol{e_i}. \end{aligned}$$

for all n points we obtain the model as follows:

$$\boldsymbol{y} = \begin{bmatrix} y1 \\ y2 \\ \vdots \\ y_n \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

and write

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{e}.$$

The elements are the vector of regression coefficients $\boldsymbol{\beta}$, the vector of observations/data/dependent variables $\boldsymbol{y}$, the vector of noise terms $\boldsymbol{e}$ and the design matrix $\boldsymbol{X}$.

In a situation with n equations and $p+1$ unknowns (more equations than unknowns), the aim is to have the "best" estimation of β, therefore there is the need for a criterion such as Sum of Squared Residuals (SSR). Assuming that b is a value for β [10]:

$$SSR(b) = ||y - Xb||^2 = \sum_{i=1}^{n} |y_i - \sum_{j=1}^{p} X_{ij} b_j|^2 \tag{1.1}$$

Now to find $\hat{\beta}$ (Minimization of SSR),

$$\hat{\boldsymbol{\beta}} = \operatorname*{argmin}_{\beta} SSR(\boldsymbol{\beta})$$

- **predictions** $\hat{y} = X\beta$.
- **residues** of residuals $\hat{e} = y - \hat{y}$.
- **estimate of $\boldsymbol{\sigma^2}$**
$$\hat{\sigma}^2 = \frac{\text{SSR}(\hat{\boldsymbol{\beta}})}{n-p} \tag{1.2}$$
- **total sum of squares** $SST = \sum_i (y_i - \bar{y})^2$
  - $\bar{y}$ is the mean value
- **residual sum of squares** $SSR = \sum_i (y_i - \hat{y}_i)^2$.
- **regression sum of squares** $SSReg = \sum_i (\hat{y}_i - \bar{\hat{y}}_i)^2$ ($\bar{\hat{y}}_i$ is the mean of predictions).
- **coefficient of determination of**
$$\mathbf{R^2} = 1 - \frac{SSR}{SST} \tag{1.3}$$

- $R^2 \in [0,1]$, $R^2 = 1$ expresses perfect fit

- **Adjusted $\mathbf{R^2}$** is used when the number of features in the model is large, R^2 increases as well even if some features do not contain any information, nor contribute to obtain the target value at all.

$$\mathbf{R^2} = 1 - \frac{n-1}{n-p}(1-R^2) \tag{1.4}$$

- **Means Squared Error (MSE)** is used as a default metric (loss function see section 1.3), for evaluation of the performance of most regression algorithms and is computed as

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{1.5}$$

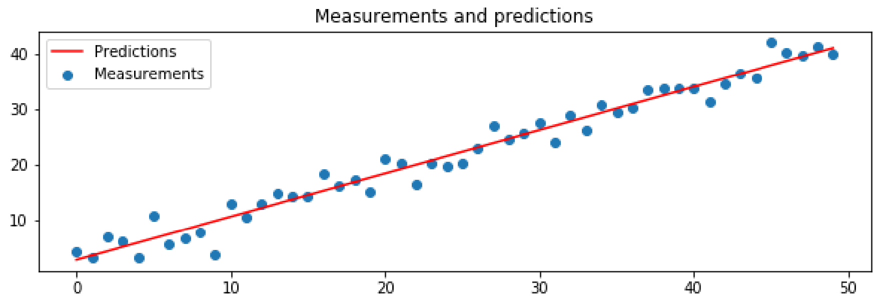


Figure 1.3: We have Measurement data points and we try to find a function that best fits the data.

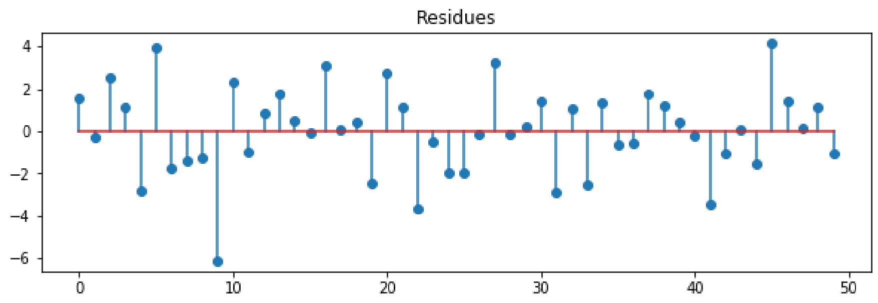


Figure 1.4: By fitting data we imply that the function is constructed in such a way that the total amount of distances from each data point from function projection is the minimum value.

1.4.2 Decision Trees

A decision tree is an algorithm that recursively divides the training data using splitting criteria and tries to predict the given target variable [11]. Similarly, as in linear regression, we want to model the relationship between the scalar response variables here denoted as Y and scalar or multivariate explanatory variable (independent variable) here denoted as X having p variables. This supervised learning algorithm wants to explain Y given $X_0, X_1, \ldots, X_{p-1}$ such that for most examples

$$Y \approx f(X_0, X_1, \ldots, X_{p-1})$$

This algorithm also shows that f doesn't need to be a mathematical function, it can be a tree. The type of function dependent on Y can be a classification problem (whether a patient is sick or healthy) or for continuous values (the price of an apartment) a regression problem. Classification trees predict an outcome that is chosen through a voting system where the majority class within a leaf vertex wins, whereas regression trees predict a numeric value that is based on the target's distribution within a vertex. A decision Tree can be used in both cases. A frequent classification problem is a binary classification, Y must consist of only two values. The input is a n-row target Y and p features $X_0, X_1, \ldots, X_{p-1}$, whereas the goal is to describe Y with a Decision tree of depth k. There exist many algorithms on how to construct the optimal tree, for example, the ID3:

1. For each element on a feature X_i calculate the value of (some suitable) criterion.

2. Select X_i with the best criterion value to Split the dataset with.

3. Continue splitting the resulting subsets until we hit the stopping rule.

Criterion can be: IG, Gini index. The stopping rule can be: selected k tree depth reached or maximum tree depth, no more attributes to be selected, no more data in the subset.

To describe how a decision tree works we will use a classification tree. Classification trees are essentially a series of questions designed to assign a classification. The image below is a classification tree trained on the IRIS dataset (flower species). Root (brown) and decision (blue) nodes contain questions that split into subnodes. The root node is just the topmost decision node.

Classification trees start at the root node (brown) and traverse the tree until you reach a leaf (terminal) node. Using the classification tree in the the image below 1.5. Imagine you had a flower with a petal length of 4.5 cm and you wanted to classify it, let's call this flower lily. Starting at the root node, the first question would be if lily's petal length (cm) is ≤ 2.45. Since lily's length is ≥ 2.45 it would follow the "false" arrow proceeding to the next decision node. On this node, the question would be if the lily's petal length (cm) is ≤ 4.95, answer being true and decision tree, lead to a leaf node, therefore could predict lily's species as versicolor.

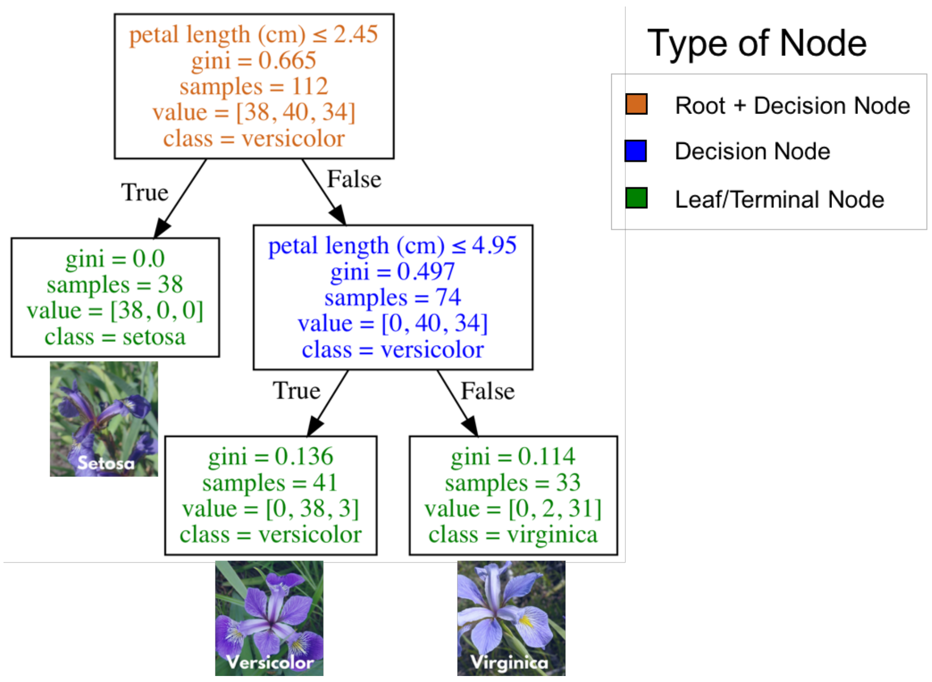


Figure 1.5: Classification tree for classification task for one of three flower species (IRIS Dataset) [12]

A classification tree learns which sequence of features to use to build the tree questions on every node. Every node splits the data into (two) branches (on each node) based on some value known as a split point, 2.45 on our case, (A) of the Figure 1.6. To understand which feature we should use to first split involves having a good split point value. A good value, which results in the largest IG, for a split point is one that separates one class from the others. An example of a good split, (B) of the Figure 1.6, all the points to the left of the split point are classified as setosa while all the points to the right of the split point are classified as versicolor.

The figure shows that setosa was correctly classified for all 38 points. It is a pure node. Classification trees don't split on pure nodes. It would result in no further IG. However, impure nodes can split further. Notice the right side on (B) of the Figure 1.7 shows that many points are miss-classified as versicolor. In other words, it contains points that are of two different classes (virginica and versicolor). If it wasn't for a stopping rule, the algorithm would continue splitting until all leaves are pure nodes, however that could lead to a very large tree depth that would lead to overfitting and usually there is a

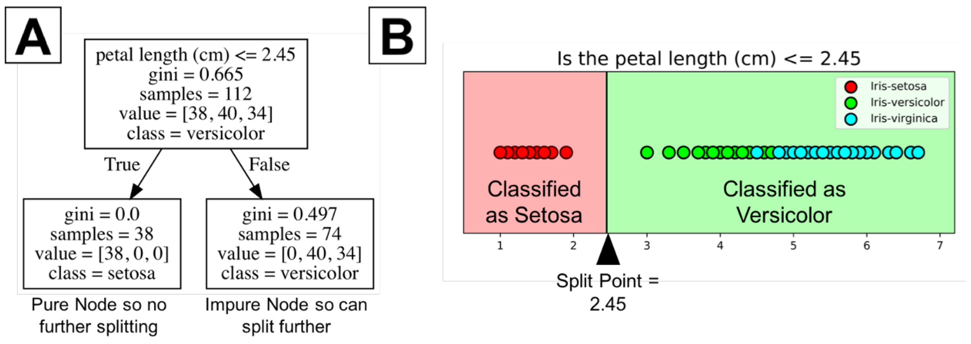


Figure 1.6: Section of classification tree explaining the goal of a split point. [12]

max tree depth specified [12]. Tree depth is a measure of how many splits a tree can make before coming to a prediction.

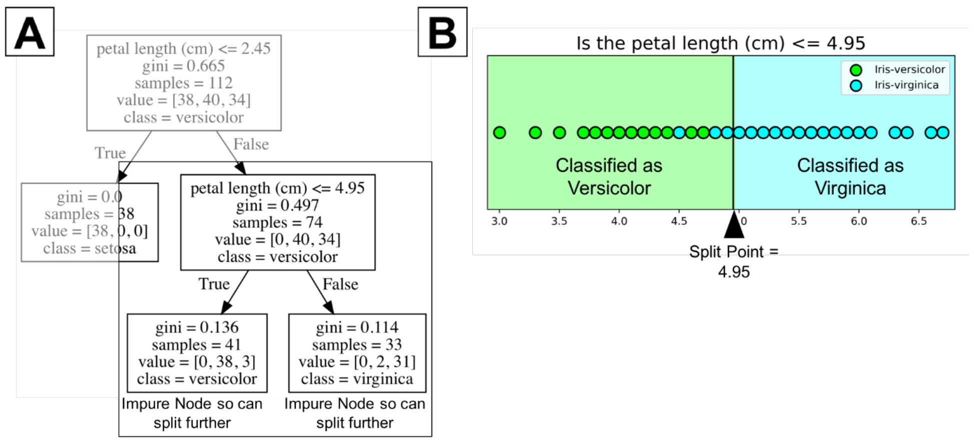


Figure 1.7: Section of classification tree explaining the goal of a split of depth 2. [12]

Decision trees split between the feature and corresponding split point that results in the largest Information Gain (IG) for a given criterion (Gini or Entropy). We want to select the feature that maximally reduces the entropy i.e. highest IG.

$$IG(D_p, X) = I(D_p) - \sum_{j \in Values(X)} \frac{|N_j|}{|N|} I(N_j) \tag{1.6}$$

Since classification trees have binary splits, split the data in two, the formula can be written as follows:

$$IG(D_p, X) = I(D_p) - \frac{|N_{left}|}{|N|} I(D_{left}) - \frac{|N_{right}|}{|N|} I(D_{right}). \tag{1.7}$$

where:

X is feature splitting upon.

D_p is dataset of the parent node.

D_{left} is dataset of left child node.

D_{left} is dataset of right child node.

I is impurity criterion Equation 1.8.

N is total number of samples.

N_{left} is total number of samples at left child node.

N_{right} is total number of samples at right child node.

Entropy is a measure that should quantify the disorder (uncertainty) of the probability. It can be zero for deterministic case (all samples at a node belong to the same class), maximal (pure randomness), increasing or decreasing:

$$I_H = -\sum_{i=1}^{c} p_i \log p_i \tag{1.8}$$

where:

p_i is the proportion of the samples that belong to class c for a particular node.

Instead of entropy once can use Gini index (Gini impurity),

$$I_G = 1 - \sum_{i=1}^{c} p_i^2 = \sum_{i=1}^{c} p_i(1 - p_i). \tag{1.9}$$

where:

p_i is the proportion of the samples that belong to class c for a particular node.

Let's assume now that we want to describe a continuous function with a Decision tree. This method can be seen as fitting a model that is piece-wise constant over some disjoint number of regions R_m,

$$f(x) = \sum_m c_m I, x \in R_m$$

and can be written as follows:

$$f(x) = c_m, x \in R_m.$$

The most common method for building a regression tree model based on a sample of unknown regression surface goes about trying to obtain the model parameters that minimise the least squares error criterion,

$$\frac{1}{n}\sum_{i}^{n}(y_i - \hat{y})^2. \tag{1.10}$$

where:

n is the sample size.

$\hat{y}$ is the prediction of the regression model for a data point $< x_i, y_i >$.

Area Under The Curve (AUC) - Receiver Operating Characteristics (ROC) is a performance measurement for classification (loss function section 1.3) problem at various thresholds settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing different classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, better the model is at distinguishing between patients with the disease and no disease [13].

1.4.3 Ensemble Techniques

There exist ML algorithms that use more than one decision tree such as Gradient Boosting Machines (GBM) and Random Forests (RF) also there is General Linear Model which includes more than one linear regression.

Tree Ensebled Techniques These algorithms are tree-based ensembles i.e. the target variable is based on more than one decision tree. GBM is a model that sequentially trains decision trees, each decision tree is built on the errors of the previous tree, additively collecting an ensemble of weak models to create a robust learning system (boosting). GBM combines two powerful tools: gradient-based optimization and boosting [14]. Gradient-based optimization uses gradient computations in training data to minimize a model's loss function. There is an initialized prediction that is used for the first tree to be built.

RF construct many individual decision trees at training, their prediction (from all the trees) are pooled–majority vote–to make the final prediction. From the original dataset, n datasets of the same cardinality are built (sampling with replacement). For each of these subsets, a small-depth decision tree is learned, each having a prediction. The final prediction of the RF is the majority vote for classification tasks or the mean for regression tasks.

Generalised Linear Model The term General Linear Model (GLM) usually refers to conventional linear regression models for a continuous response variable given continuous and/or categorical predictors. It includes more than one linear regression. GLM refers to a larger class of models where the response variable is assumed to follow an exponential family distribution, (some often nonlinear function). Some would call these "nonlinear" because of the nature of the function, however, McCullagh and Nelder consider them to be linear because the covariates affect the distribution of response variable only through the linear combination [15]. The GLM requires that the response variable follows the normal distribution whilst the generalized linear model is an extension of the general linear model that allows the specification of models whose response variable follows different distributions [16].

In "Classical" regression important assumptions are made: the outcome is a continuous variable and that it is normally distributed. "Classical" linear models, although very useful, are not suitable for many different problems. In reality, this is not always the case. Outcomes are not always normally distributed, nor are they always continuous variables. For instance, if the modeled variable is defined on $[0, 1]$, or it is binary $\{0, 1\}$, or when it denotes counts. For the GLM different link functions can be used that would denote a different relationship between the linear model and the response. for instance, logistic regression (where the dependent variable is categorical) or Poisson regression (where the dependent variable is a count variable) are both generalized linear models. GLMs are a class of models that are applied in cases where linear regression isn't applicable or fails to make appropriate predictions.

Chapter 2

Analysis of Machine Learning models

This chapter will explain the term black-box models and why they are called so, furthermore the term interpretability will be defined and elaborated based on "An introduction to Machine Learning" Interpretability[17].

2.1 Black-Box Models

Machine Learning algorithms such as Neural Networks (NN), Support Vector Machines (SVM) are considered to produce so-called black-box models because there is no direct explanation for their prediction. Since black-box models could outperform simple linear models or even decision trees their usage increases. The out-performance implying that the prediction of these models fits the data more compared to linear models, black-box models usually have a more complex relationship in the data leading to this higher predictive performance. Nevertheless, despite NN algorithms being more accurate one tends to stick to and prefer linear models due to the fact of their simplicity and are easier to interpret. Having more accurate predictive models is not always preferred since one will face the problem of interpretation. During this last century, quite the effort was and is still being made to understand the models that undergo the complex learning and making sense of the reasoning behind the predicted output. Many researchers have, therefore, developed and implemented several model-agnostic (see subsection 2.3.1) interpretability tools to have a better understanding.

Predicting modeling and ML algorithms became more complex aiming to get more accurate results, the understanding towards identifying the reasoning behind the prediction on the other hand fades away. Inscrutable ML Algorithms are called "black-box" models; such as Artificial Neural Networks (ANN), Gradient Boosting Machines (GBM), and Random Forests (RF). Even though GBM and RF are not black-box models in general, they consist of trees which actually can be explained. Due to the nature of these models, them containing a large number of trees (hundreds and more), they tend to become harder to interpret, similarly for ANN. It is exactly this complex inner structure of the ML algorithm that makes the model accurate. Due to this trade-off, data science professionals, in industries such as healthcare, insurance, banking, are being limited to using traditional, linear modeling techniques to create their predictive models.

ML is being used in companies and organizations as predictive models for a very wide variety of revenue or value-generating applications. A fraction of those applications includes deciding whether to release someone from a hospital, allowing one to make a loan. In today's data-driven commercial landscape it is crucial for the ML models to learn to their maximal potential for the companies to stay competitive. In the context of applying ML in real-life scenarios, companies tend to face a unique challenge. ML algorithms and models that they choose have to be simple and transparent enough to allow for detailed documentation of internal system mechanisms and in-depth analysis by government regulators. One must be able to explain the reasoning behind the output of the ML, therefore transparent, interpretable, and fair models are a legal mandate in banking, insurance, healthcare, and other industries. Numerous regulatory statutes are governing these industries including Civil Rights Acts, Health Insurance Portability and Accountability Act, and European Union (EU) Greater Data Privacy Regulation (GDPR) Article 22 [18]. Furthermore, regulatory statutes continuously are changing, and these regulatory regimes have a role in shaping what constitutes interpretability in applied ML.

2.2 Machine Learning Interpretability

One of the many important hopes for ML is to assist in one's day-to-day organization, aiming for simple convenience. ML also promises quick, accurate, and unbiased decision making from simple to life-changing scenarios. Theoretically, by using ML we allow the computer to make objective, data-driven decisions in crucial situations. To guarantee this promise, among other technological advances, interpretability is needed.

As demand and human nature always aims for the best results we get this fundamental trade-off. By requiring more accurate predictions, which in some cases linear models might not deliver, more complex algorithms and model structure are used which lead to losing interpretability. Interpretability is crucial to overcome the trust issues of our nature. In many industries, linear models have been preferably chosen as tools for predictive modeling of ML. The very same human nature also pushes towards closest to optimal results, nonlinear models (generated by training ML algorithms) make more accurate predictions on previously unseen data therefore are more likely to obtain such results. Linear model predictions tend to fail complex dependencies in training data which is a drawback when predicting biological data [19]. The model has to be accepted by internal validation teams and business partners and has to be approved by external regulators. In cases when non-linear models are to be used (they can allow more sophisticated and potentially more accurate decisions) interpretability can increase transparency and trust in those complex models. Using interpretable ML is fundamentally difficult and it's quite the new field of research. Complex ML techniques for the same prediction targets and set of input variables can produce multiple accurate models that are similar, however, have different internal architectures.

The models created by training ML algorithms seemed to be uninterpretable for many years, however, there have been numerous advancements made to make the often nonlinear, non-monotonic, and noncontinuous machine-learned responses functions more understandable. It is quite likely that these black-box functions will never be as interpretable and direct as more traditional linear models. The idea behind linear models is focusing on understanding and predicting average behavior, whereas more complex models can often make accurate predictions, and making it more difficult to explain subtler aspects of the modeled phenomenon. Decision trees are suitable for finding non-linear predictions and since they are interpretable the RF was designed to overcome the instability and lack of smoothness of decision trees, combining interpretability of decision trees with the performance of black-box models such as ANN. In some cases black-box models can also be simple, however, their implementation is hidden, be it in a huge amount of decision trees for RF. When a new computation is run on a network of ANN it ends up with a different new weight matrix. This is a result of starting with small random values that are then adjusted as the program runs. Aside from the random starting point a reason that it is difficult to describe, mathematically at least, a network is in a state of change, and mathematics are in effect stateless. That is we can use mathematics to describe a start, an end, and a point in the process, but the learning process is a sequence of results from many operations and is not stateless. This is why ANN's are black-box [20].

Let's say we want to predict the number of purchases knowing a customer's age. On Figure 2.1 we model the relationship between those factors and obtaining a function $g(x)$, which fails to model youngsters purchasing more (lost profits) than older generations. Whereas On Figure 2.2 which uses a more complex non-linear model is more successful in modeling this relationship.

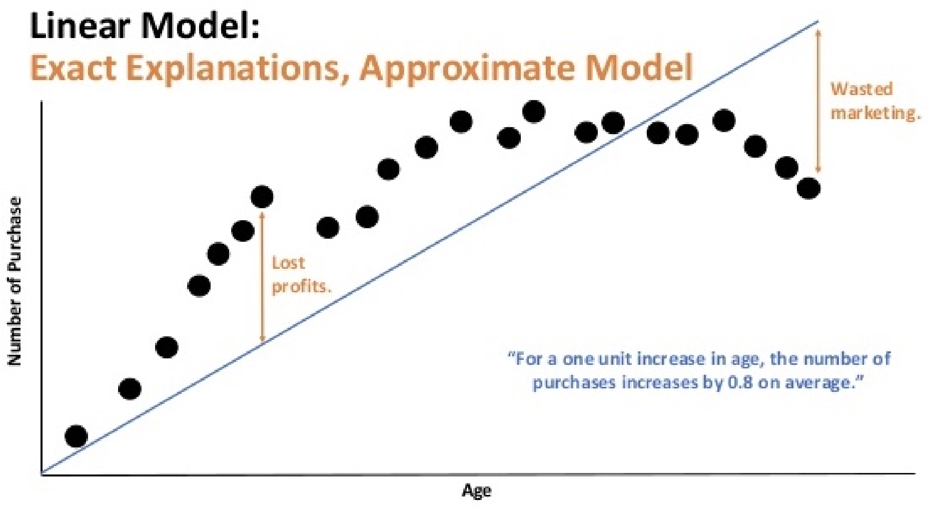


Figure 2.1: Linear model $g(x)$ given a customer's age predicts the average number of purchases. Prediction explanations are straightforward and stable, however can be inaccurate [21].

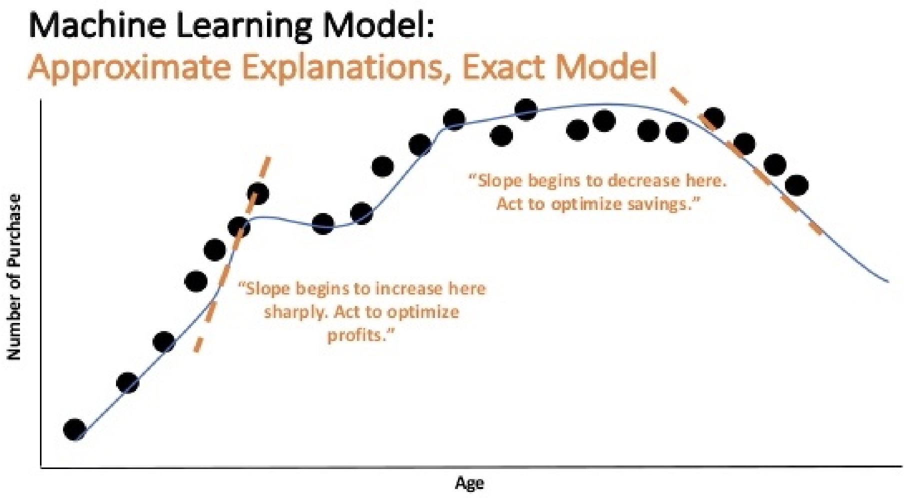


Figure 2.2: Linear model $g(x)$ given a customer's age predicts the average number of purchases. Prediction is very accurate, almost replicating the actual, unknown generating function, $f(x)$ [22].

Interpretability, in the context of ML models and their results, has been defined as "the ability to explain (present) in understandable terms to a human". The complexity of a ML algorithm model often has a high relation with interpretability. Hence, the more complex the ML model the more one loses interpretation and explanation. Upon analyzing the functional form of a model and discussing their degree of interpretability these various class cases follow [17]:

High Interpretability This class refers to linear and monotonic models. In Linear and monotonic models by a change in any given input variable (or combination or function of an input variable), the output of the response function does change. The change is with a defined rate, in only one direction, at a magnitude that can be represented by a coefficient. Monotonicity lets one have an intuitive, and in some cases even automatic, reasoning about predictions. Models in this class are mostly composed of traditional regression algorithms.

Medium Interpretability This class refers to nonlinear, monotonic models. "Although most machine-learned response functions are nonlinear, some can be constrained to be monotonic with respect to any given independent variable" [17]. By changing a single input variable in nonlinear and monotonic functions there is no single coefficient that represents the change in the response function, however, it is known they always change in one direction. Therefore, nonlinear and monotonic response functions are interpretable and suitable for use.

Low Interpretability This class refers to nonlinear, nonmonotonic models, which includes most ML algorithms. In this class the functions change in both positive and negative directions with an unstable rate for a change in an input variable, therefore this is the least interpretable class. The (only) standard interpretability measures provided by these functions are the feature importance measures.

2.3 Feature Importance

Since ML algorithms such as NN often produce black-box models losing interpretability, however, outperforming linear models or decision trees in predictive performance as they tend to model complex relationships in the data. Many types of research have therefore developed various interpretability tools, which visualize or express the unprovided explanation of the black-box model prediction by feature importance. Feature importance describes how or to what extent each feature contributes to the prediction of the model. The feature contribution and feature attribution can be calculated either on a local or a global level.

We use the term feature importance to describe how important the feature was for the predictive performance of the model, regardless of the shape (e.g., linear or nonlinear relationship) or direction of the feature effect. The higher the value the more important the feature. This implies that measures of feature importance require knowledge of the true values of the target variable. The most prominent approach is the permutation importance introduced by Breiman [23] for random forests [24]. "Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node" [25]. To calculate the node probability we simply divide the number of samples reaching the node by the total number of samples. For each decision tree we calculate (assuming binary tree) a single node's importance using Gini Index 1.9 or IG 1.6 for classification tasks and Mean Square Error for regression tasks 1.10,

$$ni_j = w_j I_j - w_{lj} I_{lj} - w_{rj} I_{rj}.$$

where:

ni_j is j's node importance (importance of node j).

w_j weighted number of samples that reached node j (subscript "lj" and "rj" imply child node from left and right split on node j, respectively).

I_j is the Impurity value on node j (similarly, sub "lj" and "rj" implication is observed).

Then, for each feature on a decision tree the importance is calculated as following:

$$fi_i = \frac{\sum_j ni_j}{\sum_{n \in N} ni_n}.$$

where:

fi_i feature i-th importance.

ni_j importance of node j (j: node j splits on feature i).

n is a node from all the nodes N.

To get normalized values between 0 and 1,

$$fi_{i(norm.)} = \frac{fi_i}{\sum_{j \in allfeatures} fi_j}.$$

Finally, to calculate feature importance values from each tree (for Random Forest),

$$RFfi_i = \frac{\sum_j fi_{ij(norm.)}}{\sum_{(j \in allfeatures)(k \in alltrees)} fi_{ik(norm.)}}.$$

In linear regression models the importance of a feature can be calculated by the absolute value of the estimated weight scaled with its standard error 1.2. The linear regression model predicts the target as a weighed sum of the feature inputs. The importance of a feature increases with increasing weight, the more variance the estimated weight has (lower certainty about the correct value), the less important the feature is. The importance of a feature is in inverse proportion with the estimated weight of a feature 1.1 ($\hat{\beta}$ representing learned feature weights):

$$t_{\hat{\beta}_j} = \frac{\hat{\beta}_j}{SSR(\hat{\beta}_j)}.$$

Let's consider weather and calendar information, and let's predict the number of rented bikes, data from [26], for a particular day (we examine the estimated regression weights here).

Features	Weight	SSR	$\|t\|$
...	...	...	...
weathersitMISTY	-379.4	87.6	4.3
weathersitRAIN/SNOW/STORM	-1901.5	223.6	8.5
temp	110.7	7.0	15.7
...	...	...	...

Table 2.1: Feature importance for linear regression subsection of dataset.

The table consists of numerical and categorical features, and shows each feature estimated weight, SSR, and the absolute value of the t-statistic. Interpretations for features follows [27]:

- Numerical feature "temperature": By an increase of temperature by one degree Celsius the target variable increases by 110.7 and other features remain fixed.
- Categorical feature "weathersit": On days when it is raining, snowing or stormy, the estimated number of bicycles drops by 1901.5 (again, the other features remain fixed) and drop by 379.4 when weather is misty, that is compared to the number of bikes on good weather.

The advantage of "isolating" other features on linear regression–where the predicted target is a linear combination of the weighted features–is that it directly gives an insight into the increase or decrease of a certain feature to the prediction. The disadvantage is that the interpretation ignores the joint distribution of the features. In some realistic scenarios, the change of a feature can lead to affect another (increasing the number of rooms in a flat increases the total number of the area in the flat).

2.3.1 Model-agnostic Interpretability

It is vital to be able to explain the ML predictions by providing a rationale using textual, visual components to gain knowledge of what would happen if the components were different. Instead of supporting models that are labeled as black-box models, a prevailing solution was to only use "interpretable" models. These approaches use models in which one can derive the model's components directly. "An alternative approach to interpretability in ML is to be model-agnostic, i.e. to extract post-hoc explanations by treating the original model as a black-box " [28]. By learning an interpretable model on the predictions of the black-box model, changing the input in certain ways to observe how the model reacts. Since linear models are inherently crippled (in the sense of most accurate predictions deliverance) by the need to be interpretable they are not of the highest level suited for most real-world applications. The separation of interpretability from the model thus frees up the model to be as flexible as necessary for the task, enabling the use of any ML approach also allowing the control of the complexity-interpretability trade-off which is defined as following [28, 29]:

Model flexibility and interpretation: "One cannot use a model whose behavior is very complex, yet expect humans to fully comprehend it globally". The paper argues that separating the interpretability from the model frees up the model to be as flexible as needed such as for arbitrary Deep NN. This allows the interpretation method to work with any ML model.

Explanation flexibility: Different explanations are needed in different situations, such as how the model would behave if certain features had different values. One is not limited to a certain explanation. This doesn't bound the model to be always black-box, certain cases seem to have a linear formula as more fit. By keeping the model separate from the explanations one can obtain the information needed whilst the model is fixed. Therefore, not being limited to a certain form of explanation.

Representation flexibility: In the cases of the features them-self being not interpretable, the interpretable model can be created on such features, however not being interpretable persists. Model-agnostic approaches can be used to generate explanations by using different features as opposed to the underlying model. The explanation system can be able to, instead of using the non-interpretable features, use different feature representations.

The reasoning behind having the explanation of feature significance separately from the ML model (model-agnostic interpretation methods over model-specific ones) is that their flexibility lets one choose ML algorithm more freely. In terms of interpretability, it is more suited to work with mode-agnostic explanations since the same method can be used for any type of model. Model-Agnostic explanation systems explained in Riberio, Singh, and Guestrin work [29].

2.4 Global and Local Interpretability

To understand the entire model one has trained on a global scale, global interpretations help to understand the inputs and their relationship with the prediction target. It is also useful to zoom in into local parts of the data or predictions to derive local explanations.

Global model interpretation is a set of techniques that try to translate the model's behavior in general. Finding such features that drive predictions and such features that are more or completely useless for the prediction. This knowledge helps explain the model, this knowledge being the relationship between the target variable and the other variables. Most global interpretation techniques get this knowledge by investigating the conditional interactions between the dependent variable and the independent variables on the complete dataset.

Local model interpretation is understanding small regions of the machine-learned relationship between the input variables and the target variable. Local interpretability allows one to answer why the model made a certain prediction for an instance, by zooming in on that instance (or a cluster of input records) and it's (their) corresponding prediction(s).

Local interpretations are focused on single features or a group of similar features that lead to the model prediction. Local explanations tend to be more accurate than global ones because small regions of a machine-learned response function are more likely to be linear and monotonic. By combining results of both global and local interpretations techniques is the way to get the best explanation results [17].

Chapter 3

Sensitivity Analysis

It is impossible to define a single set of rules that would apply to every modeling. If one characterizes modeling as a heterogeneity lacking systematization then sensitivity analysis would seem overly ambitious to offer an universal application [30]. In an economics background, one derives instances of sensitivities as outputs versus a particular input. On the other hand, others who use sensitivity analysis in practice, which are fewer in number, have a different view on this matter. This minority actively uses importance measures that have a more global approach, that is, assessing the output factors by analyzing the entire input space, rather than locally looking at a point in that space.

3.1 Global Sensitiviy Analysis

A simple yet powerful way to understand a ML model is by doing sensitivity analysis where one examines the impact of each feature on the model's prediction. Global Sensitivity Analysis as stated by Salteli [30] is the study of how a model's uncertain output (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input. It determines how the output (dependent variable) is affected by different values of input (independent variables) under a given set of assumptions. Sensitivity Analysis studies how a couple of different sources of uncertainty, mathematically add to the overalls models uncertainty. In simple terms, one will look at how the variables move as well as how the target is affected by the input variables, which can determine the most contributing features on output behavior.

There are numerous applications to Sensitivity Analysis such as model verification, understanding and simplifying the model. Sensitivity Analysis first appeared as the local approach, when observing the impact of small input perturbations on the model and the output is studied. Local methods have their limitations, therefore throughout the years a new class of methods has been developed and is referred to as Global Sensitivity Analysis. It is referred to as global since it considers the entire variation range of the independent variables. Sateli and Pappenberger [30] emphasized the need to specify the objectives of a study clearly, before making a sensitivity analysis. Among the objectives, there could be a need to identify and prioritize the most influential inputs (the random variables that mostly affect the prediction), or the non-influential ones. One can fix the non-influential inputs to nominal values (the mean of the random variable for instance). An objective could also be to focus only on a specific domain of inputs if necessary. On parametric models (models that include fixed numeric values) one can assume their values to make the model more flexible. “Parameter values can be guessed by expert knowledge, therefore even avoiding the need for observed data” [31].

To calculate the sensitivity of a feature one changes the feature’s values (or even ignore it), while other features stay constant, and observe the output. If the model’s outcome changes drastically by changing the feature’s value, it means that the feature changed has a big impact on the prediction. The higher the change rate of the output’s model the more significant a feature’s role on the output. “A good sensitivity analysis should conduct analyses over the full range of plausible values of key parameters and their interactions, to assess how impacts change in response to change in key parameters” [32]. Since the models can be made to produce virtually any desired behavior with both plausible structure and parameter values it is considered that a quality check, provided by a careful sensitivity analysis, it’s worth the effort. Based on the work and experience of the European Commission [32] it is believed that the target of interest shouldn’t be only the model’s output, the target should also be the question itself that the model has been called to answer. To give a better understanding an example was given: “One should seek from the analyses conclusions of relevance to the question put to the model, as opposed to relevant to the model” [30]. To clarify, the output of interest applies to the usage of the model, not the model building itself.

There are screening qualitative methods, called screening methods, which allow studying sensitivity measures of importance. Screening methods are based on a discretization of the inputs in levels, allowing a fast exploration of the code behavior.

3.2 Permutation Feature Importance

“Permutation Feature Importance (PFI) measures the increase in the prediction error of the model after we permuted the feature’s values, which breaks the relationship between the feature and the true outcome”[33]. After building the model which predicted the target variable PFI is measured by calculating the increase in the model’s prediction error after permuting a feature of the dataset. Let X_j be a feature in the dataset and we want to measure its importance. By shuffling its values we create X_j^p. Upon scoring (predicting) the target variable with the permuted feature X_j^p instead of X_j can lead to an increase of the model's error, then feature X_i is considered important.

PFI evaluates how much the models prediction relies on each feature. Let's refer to the input dataset as $Z = [yX]$, and assume a matrix composed of a n-length with target vector $\boldsymbol{y}$ as the first column , followed by remaining columns as $n \times p$ matrix $X = [X_0, \ldots, X_p]$.

```
Algorithm 1: Permutation Feature Importance
  Result: Sorted array of $FI$
  Input: Trained model $f$, dataset $Z$, error measure $L(y, f)$.
  Output: Sorted features $FI$
  Calculate original model error $e_{og} = L(y, f(X))$ (eg. AUC, MSE);
  for $i \leftarrow 0$ to $p$ do
      Randomly shuffle $X_i$ into $X_i^p$, $X_i \in X$ ;
      Replace $X_i$ with $X_i^p$ denoted as $X^p$, $X_i^p \in X^p$;
      Calculate estimation error $e_i^p = L(y, f(X^p))$;
      Calculate $FI_i = e_i^p - e^{og}$;
      Replace $X_i^p$ with $X_i$;
  end
  Sort features $FI$;
```

If the FI_j value with permutated feature X_j leads to no increase or a low one then the feature is not important or less important. This approach was first introduced by Breiman [23] for RF, then later on Fisher, Rudin, and Dominici [34] built upon Breiman's idea and proposed a model-agnostic version which was called Model Reliance on their paper.

3.3 One At a Time

One At a Time (OAT) is a screening method in which each input is varied while fixing the others. Screening methods are based on a discretization (the process through which one can transform continuous variables or models into a discrete form) of the inputs in levels, allowing quick exploration of the behavior. Based on practice it is shown that only a small number of inputs are influential. The aim of screening techniques is to identify the non-influential inputs, without the need to have a lot of model calls, making realistic hypothesis on the model complexity [35].

To implement the Morris Screening method, a number r of different trajectories through variable space have to be constructed. When the number of trajectories r is small, it is possible that not all the possible factor levels are explored. It is assumed that valuable results can be obtained for r in the range of 4 to 10. Indices are obtained as follows [35]:

$$\mu_j^* = \frac{1}{r} \sum_{i=1}^{r} |FI_j^{(i)}| \tag{3.1}$$

$$\sigma_j = \sqrt{\frac{1}{r}\sum_{i=1}^{r}\left(FI_j^{(i)} - \frac{1}{r}\sum_{i=1}^{r} FI_j^{(i)}\right)^2} \tag{3.2}$$

where:

FI is Feature Importance calculated on algorithm 1.

μ_j^* is a measure of the influence of the j-th input on the output (mean of the absolute value of the feature importance). The larger μ_j^* the more the j-th input contributes to the dispersion of the output.

σ_j is a measure of the non-linear and/or interaction effects of the j-th input (standard deviation of the permutation feature importance). If σ_j is small, elementary effects have low variations on the support of the input, suggesting a linear relationship between the studied input and the output. Whereas the larger the σ_j the less likely the linearity hypothesis is. Thus a variable with a large σ_j will be considered having non-linear effects or implies interaction with at least one other variable.

The method of Morris allows to classify the features into three groups: features having negligible effects, features having large linear effects without interactions, and features having large non-linear and/or interaction effects. The method consists of randomly shuffling the input space for each feature one at a time, then performing a given number of OAT designs. The repetition of these steps allows the estimation of elementary effects for each feature.

3.4 Permutation Feature Importance for Model selection

Keeping in mind that a ML models goal is to generate the real-world scenarios and model the reality for predictions. Permutation Feature Importance, for example in RF, measures the decrease in prediction accuracy depending on the information on each independent features, or in NN where noise is added to independent features. This existing Feature Importance does not generally account for the fact that there can be many models constructed to fit the data (almost) equally well, meaning that different models may rely on entirely different independent features for predicting outcomes. This scenario is known as the “Rashomon” effect of statistics. The Rashomon effect concerns how to give comprehensive descriptions of the input features, also raises the question does the model with the best predictions necessarily gives the most accurate interpretation? [34] That is not the goal of this thesis, however, the realization can be used in that field.

Chapter 4

Implementation

The main parts of Implementation are:

- Used libraries and technologies,
- Permutation Feature Importance
- Permuting the Dataset,

4.1 Introduction to H2O

H2O.AI is a technology involved with impressive force in AI and ML. Based on Ellen Friendman [36] it is practiced across a wide range of business settings, with options that work well for both beginnings, less experienced teams, as well as more professionals. Providing very powerful and sophisticated approaches that advanced data scientists look for. H2O.AI has both free and open-source offerings as well as a choice of enterprise-grade products and services. H2O-3 is a highly scalable, distributed, in-memory, very fast open-source software technology for building AI and ML systems. It can run on-premises or in the cloud and provides production-ready artifacts for deployment. H2O-3 requires Java but also lets you use familiar programming languages, including Python, R, or Scala to build ML models for both supervised and unsupervised approaches. H2O-3's REST API allows access to all the capabilities of H2O-3 from an external program or script via JSON over HTTP. The Rest API is used by H2O-3's web interface (Flow UI), R binding (H2O-R), and Python binding (H2O-Python). H2O-3 has numerous ML algorithms developed which one can use for building models.

4.2 Implementation of Permutation Feature Importance

My implementation of PFI will be used after training and scoring the model which will give a tabular form of features with the values of influence on the outcome. The model is created and after training so that it can be used to score (make a prediction). I will be using CSV files to read the dataset. A CSV is a comma-separated values file, which allows data to be saved in a tabular format. In code the naming changed from Permutation Feature Importance to Permutation Variable Importance. I will read the CSV file, and the dataset will be represented as a Frame within H2O-3. The features are represented in Vec (vectors) containing the values of the features. I implement PFI as it was presented in the PFI chapter algorithm 1. There are different prediction categories for models, such as Binomial, Multinomial, Regression, etc. implying that not every model can have all the metrics pre-calculated. Since I am allowing the user to select the metric to be used to find PFI, I shall first show the pre-processes of the algorithm. As stated in the algorithm as input we will have the trained model m, dataset Z which from now on I will denoting as a Frame, and error measure $L(y, m)$ which from now on will be referred to as a metric. Let's start decomposing the lines of code, and how it is implemented. PFI is obtained on the following steps:

1. Preprocess: First lines of the code are regarding setting the variables to shuffle and setting model training metric. Method "arrangeColsToShuffle()" collects in a list the response column (target variable) and (if any) variables to be ignored which can be set by the user. Furthermore, users can define weight and fold columns as a part of a frame and these columns should not be treated as features. Method "setOgModelMetric()" sets the metric selected by the user. In H2O-3 when a model is trained it also generates the metrics based on the algorithm specified and tasks specified. I run checks to see which category the model belongs to ensuring the selected metric exists for this model, store it, otherwise throw an error. Also, it is stated that MSE exists for every model in H2O-3 so I use it as a default (if the user doesn't specify the metric) see Equation 1.5.

2. I will go through all the variables to be shuffled which is a subset of the Frame Z, since it's not including the List of variables already set.

```
1  public TwoDimTable permutationVarImportance(){
2  arrangeColsToShuffle();
3  setOgMetric(); // get the metric value from the model
4  _varImpMap = new HashMap<>(_varsToShuffle.length);
5
6  int id = 0;
7  for (int f = 0; f < _inputFrame.numCols() ; f++){
8      if (isIgnored(_variables[f]))
9          continue;
10     // shuffle values of feature
11     Vec shuffledFeature = VecUtils.ShuffleVec(
          _inputFrame.vec(_variables[f]), _inputFrame.vec(
          _variables[f]).makeCopy());
12     Vec ogFeature = _inputFrame.replace(f,
          shuffledFeature);
13
14     // score the model
15     Frame newScore = _model.score(_inputFrame);
16
17     // set and add new metrics ~ fills @param _p_var_imp
           needed for ModelMetrics.calcVarImp()
18     setVariablesMetric(id);
19
20     // return the original data and add to map
21     _inputFrame.replace(f, ogFeature);
22     _varImpMap.put(_variables[f], _pVarImp[id++]);
23
24     newScore.remove();
25     shuffledFeature.remove();
26 }
27 // Create TwoDimTable having (Relative + Scaled +
       percentage) importance
28 _permutationVarImp = ModelMetrics.calcVarImp(_pVarImp,
       _varsToShuffle);
29 return _permutationVarImp;
30 }
```

3. Now I shuffle feature X_i and create X_i^p (“shuffledFeature”) (line 11). The shuffling is done according to Fisher-Yates algorithm [37], To see the shuffling mechanism in more detail see number 7. The replace method, updates the vector at position i with the shuffled Vec and returns the original Vec denoted as ogVar, updating the Frame.

4. now I call the scoring method from the model (line 15) which scores the model with the Frame which has the variable shuffled. The scoring method will also generate new metrics which I am going to extract and store (line 18). Similarly as “setOgModelMetric()”, “setVariable-

Metric()" retrieves the metric from the model according to the specified metric. Also, calculating $FI_i = e_i^p - e^{og}$ and storing it on an array, which will be used to generate a two dimensional table (line 28).

5. I return the Frame to its original form, replacing the shuffled Vec with the original Vec (line 21). I map variables FI_i score it's i name (String) to the score (line 22). Remove the Frame and Vec which are no longer useful.

6. After every variable gets the FI_i score, we sort the importance. The Sorting takes place in the "ModelMetrics.calcVarImp(...)" method which also creates a data structure called a TwoDimTable, which is a two-dimensional table, which has row names as well. For the header of columns I leave the variables names and for the rows Relative importance, Scaled importance, and Percentage.

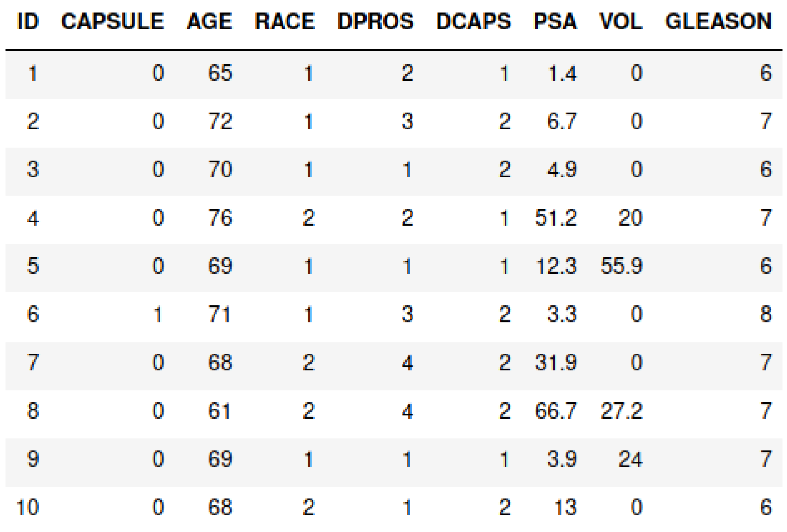
7. The Fisher-Yates algorithm is implemented within H2O-3:

```java
public static class ShuffleVecTask extends MRTask<ShuffleVecTask> {
@Override public void map(Chunk ic, Chunk nc) {
  Random rng = getRNG(seed(ic.cidx()));
  for (int i = 1; i < ic._len ; i++) {
    // inclusive upper bound <0,i>
    int j = rng.nextInt(i);
    switch (ic.vec().get_type()) {
      case Vec.T_BAD: break; /* NOP */
      case Vec.T_UUID:
        if (j != i) nc.setAny(i, ic.at16l(j));
        nc.setAny(j, ic.at16l(i));
        break;
      case Vec.T_STR:
        if (j != i) nc.setAny(i, ic.stringAt(j));
        nc.setAny(j, ic.stringAt(i));
        break;
      case Vec.T_NUM: /* fallthrough */
      case Vec.T_CAT:
      case Vec.T_TIME:
        if (j != i) nc.setAny(i, ic.atd(j));
        nc.setAny(j, ic.atd(i));
        break;
      default:
        throw new IllegalArgumentException("Unsupported vector type: " + ic.vec().get_type());
    }
  }
}
}
```

The Map/Reduce style distributed system (MRTask) includes a map and reduce methods that can be overridden to specify a computation. MRTask is used to distribute the computation over threads and machines. The map overload gives a single local input Chunk. Chunks (a compression scheme, over a chunk of data) are collections of elements and support an array-like API. Chunks are subsets of Vecs. Chunks are mapped many-to-one Vecs. Chunk updates are not multi-thread safe, therefore I use map/reduce jobs on a single column in an input Frame. Chunks hold Java primitive values, timestamps, UUIDs, or Strings. This method extended from MRTask parent class (line 1) overloads the map method and is given a two local input Chunk (line 2). Since I am using the Fisher-Yates algorithm, which consists of accessing a random element of, in this case, the chunk as input, I give the source Vec with the original values and a copy of it which will be shuffled and returned. I found it more convenient and straightforward to make a copy than to initiate a new Chunk with NaN values and with the Vec type and size accordingly. The map method takes two parameters Chunk ic and Chunk oc. When dealing with Larger Vecs, it is separated into some Chunks. Chunk *ic* will refer to the input Chunk (original values), whereas Chunk *oc* will refer to the output Chunk with shuffled values. The Chunks in the MRTask are separately shuffled then during the reduce phase it put them together in a Vec which is returned.

The input dataset which I will be using to demonstrate PFI is prostate.csv. The file is read and parsed into a Frame. The dataset file is taken from the smalldata folder in h2o-3 repository, and is the following:

Figure 4.1: First 10 rows of Frame prostate.csv



ID	CAPSULE	AGE	RACE	DPROS	DCAPS	PSA	VOL	GLEASON
1	0	65	1	2	1	1.4	0	6
2	0	72	1	3	2	6.7	0	7
3	0	70	1	1	2	4.9	0	6
4	0	76	2	2	1	51.2	20	7
5	0	69	1	1	1	12.3	55.9	6
6	1	71	1	3	2	3.3	0	8
7	0	68	2	4	2	31.9	0	7
8	0	61	2	4	2	66.7	27.2	7
9	0	69	1	1	1	3.9	24	7
10	0	68	2	1	2	13	0	6

Whether to calculate PFI on test data or train data, depends on what one is seeking to obtain from the result. Upon using training data it shows how much the model relies on each variable for making the prediction, whereas upon using testing data it shows the performance of the model on unseen data [33]. I am focused on the reliance of each variable towards the prediction, therefore will be using the training data. The first model I use to do the prediction is GBM model, see subsection 1.4.3. GBM by design has feature importance calculated and I can use those results and compare them to PFI. The core implementation is in Java, however, the user can use the implementation in Python and R. The following plots can be executed by the user within h2o-3 with Python.

Figure 4.2: Result of Permutation Feature Importance for prostate Frame, Scaled Importance using metric logloss

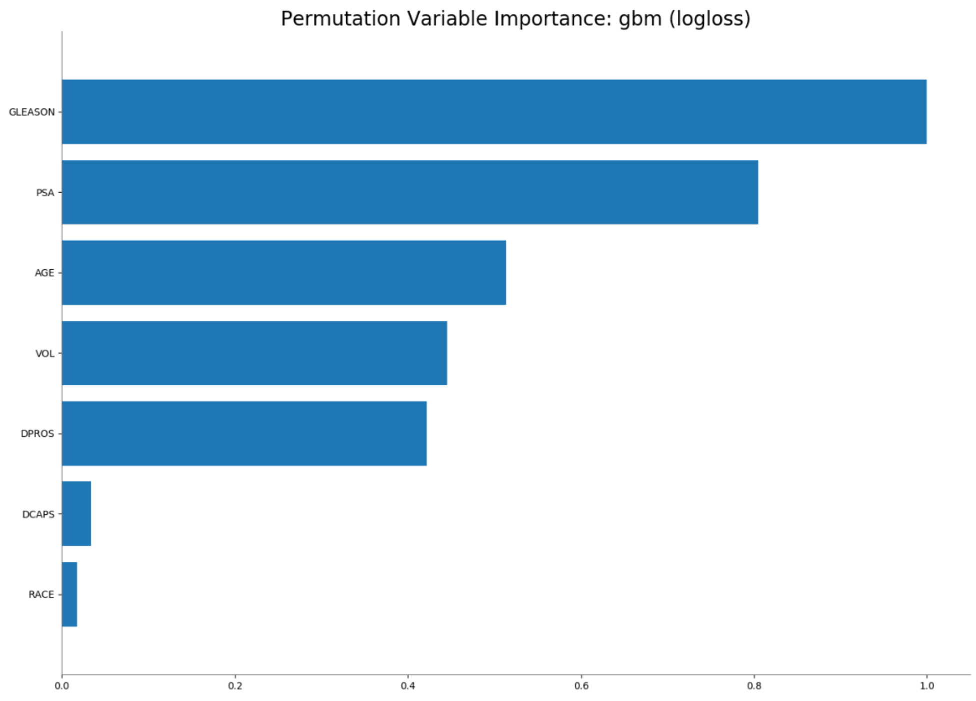
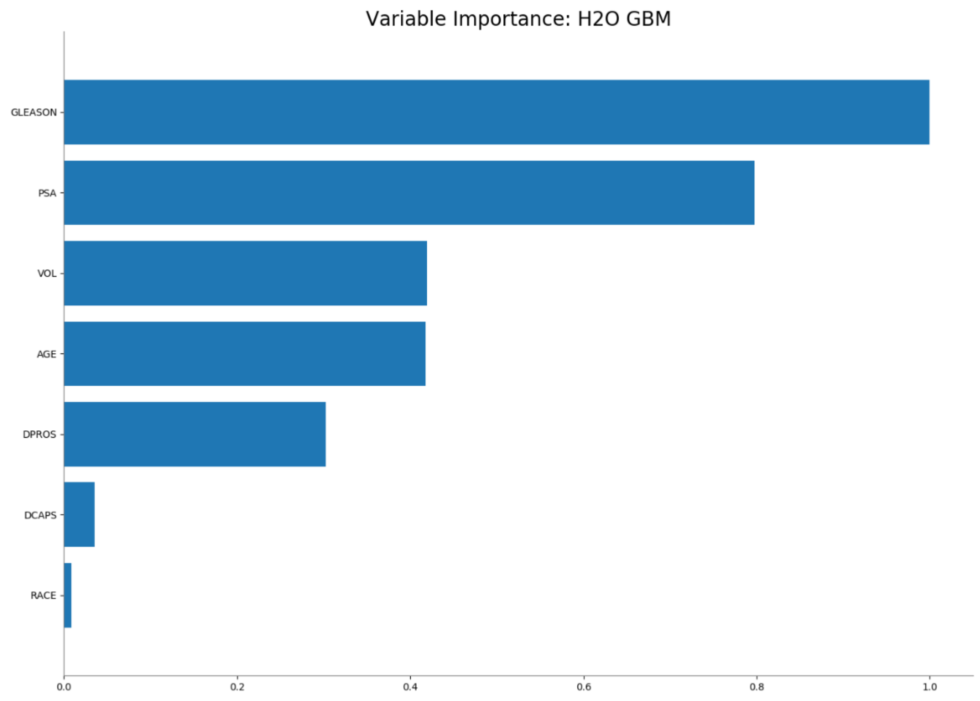


Figure 4.3: Feature importance within GBM



4.2.1 Implementation of One At a Time screening technique

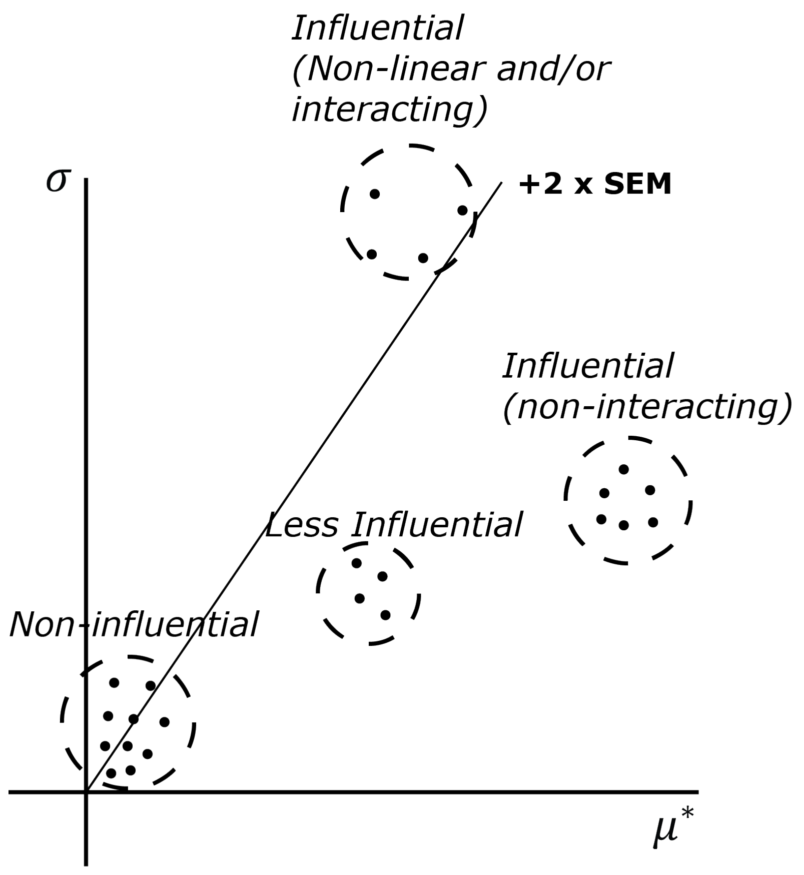
The Morris method for Global Sensitivity Analysis is a so-called One At a Time (OAT) method, meaning that in each run only one input parameter is given a new value. It facilitates a Global Sensitivity Analysis by making a number r of local changes at different input points of the possible range of input values. The local changes made on the Frame are as explained in PFI algorithm algorithm 1 the change of input points being randomly selected and swapped within the column. Now we will use the OAT screening technique see section 3.3 to explain which features have the highest importance. The literature suggests r as a number between 4 and 10 [35].

1. We execute "permutationVarImportance()" r times and use this information to calculate the mean of the absolute value of the PFI of every feature μ^* see Equation 3.1 and the standard deviation of the feature importance of every feature σ see Equation 3.2.

2. Now based on these two we can analyze and categorize the importance of the features to the output of the model.

To illustrate which features are influential and which not see Figure 4.4

Figure 4.4: parameter importance ranking



Now that I have shown how to interpret μ^* and σ let's execute all the steps mentioned in the chapter on a real dataset, prostate.csv. We build and train a GBM model, to predict "CAPSULE". For *FI* scores see Figure 4.2, Morris OAT method see Figure 4.5: Input "GLEASON" is the most influential group, whereas "VOL", "DROPS", "PSA" are on influential group, then "DCAPS" and "DROPS" as non-influential.

Figure 4.5: Result of Morris method for $r = 4$ on prostate Frame execution 1

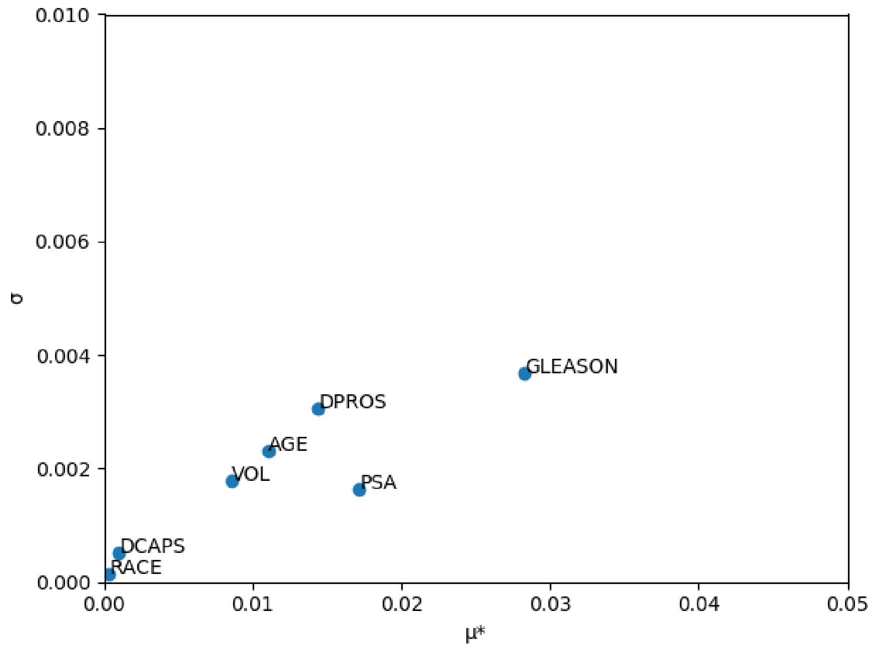
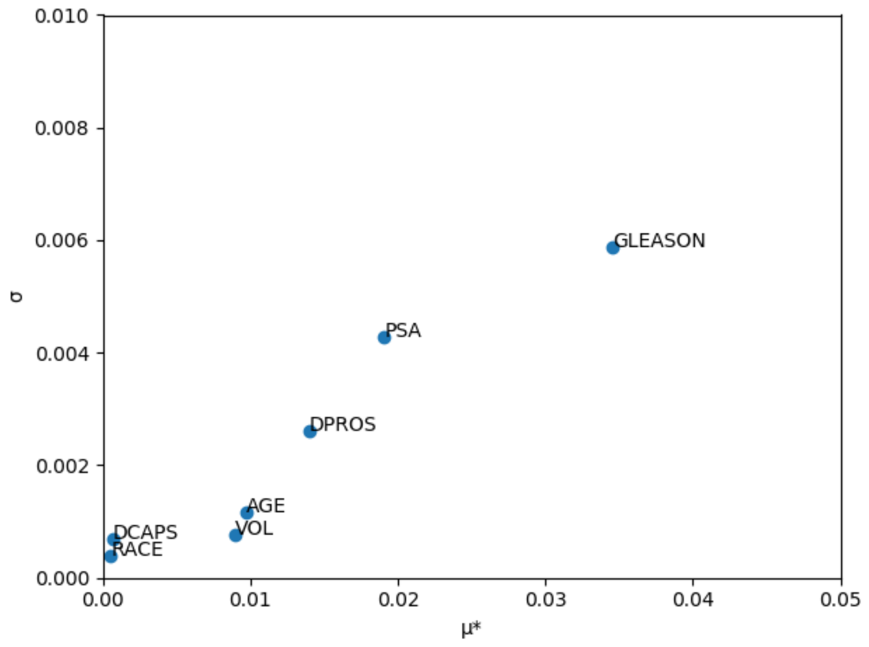


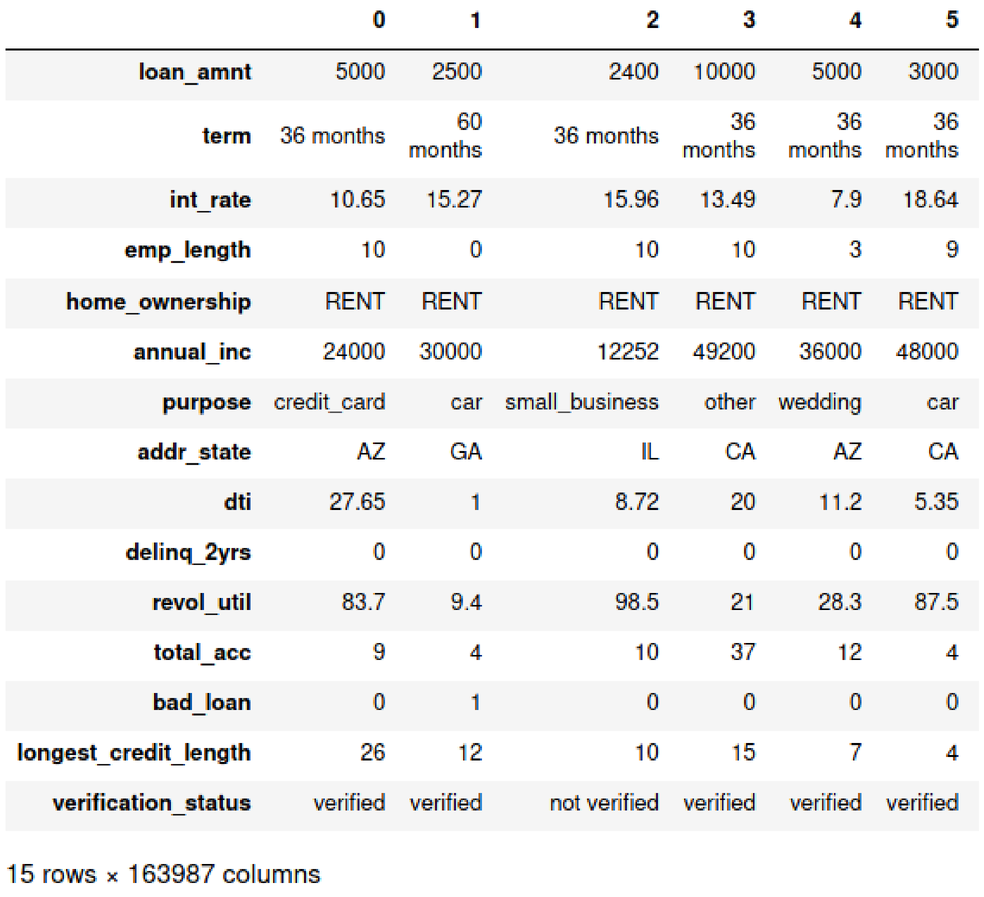
Figure 4.6: Result of Morris method for $r = 4$ on prostate Frame execution 2



4.2.2 Testing with big data

The following dataset has 15 columns and 163987 rows. The response column shall be “bad loan”, whether the loan will be repaid or not. The dataset is within the big data folder of h2o-3 libraries.

Figure 4.7: The first rows (transposed for visibility purposes) of the loan dataset



	0	1	2	3	4	5
loan_amnt	5000	2500	2400	10000	5000	3000
term	36 months	60 months	36 months	36 months	36 months	36 months
int_rate	10.65	15.27	15.96	13.49	7.9	18.64
emp_length	10	0	10	10	3	9
home_ownership	RENT	RENT	RENT	RENT	RENT	RENT
annual_inc	24000	30000	12252	49200	36000	48000
purpose	credit_card	car	small_business	other	wedding	car
addr_state	AZ	GA	IL	CA	AZ	CA
dti	27.65	1	8.72	20	11.2	5.35
delinq_2yrs	0	0	0	0	0	0
revol_util	83.7	9.4	98.5	21	28.3	87.5
total_acc	9	4	10	37	12	4
bad_loan	0	1	0	0	0	0
longest_credit_length	26	12	10	15	7	4
verification_status	verified	verified	not verified	verified	verified	verified

15 rows × 163987 columns

Figure 4.8: Showing 10 most influential variables of PFI using MSE metric

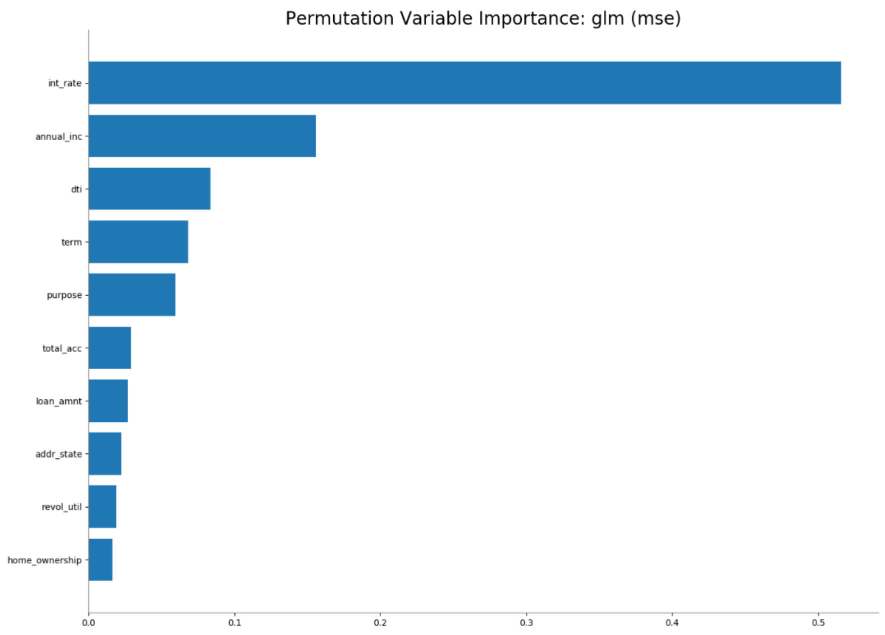


Figure 4.9: Showing 10 most influential variables of PFI using log-loss metric

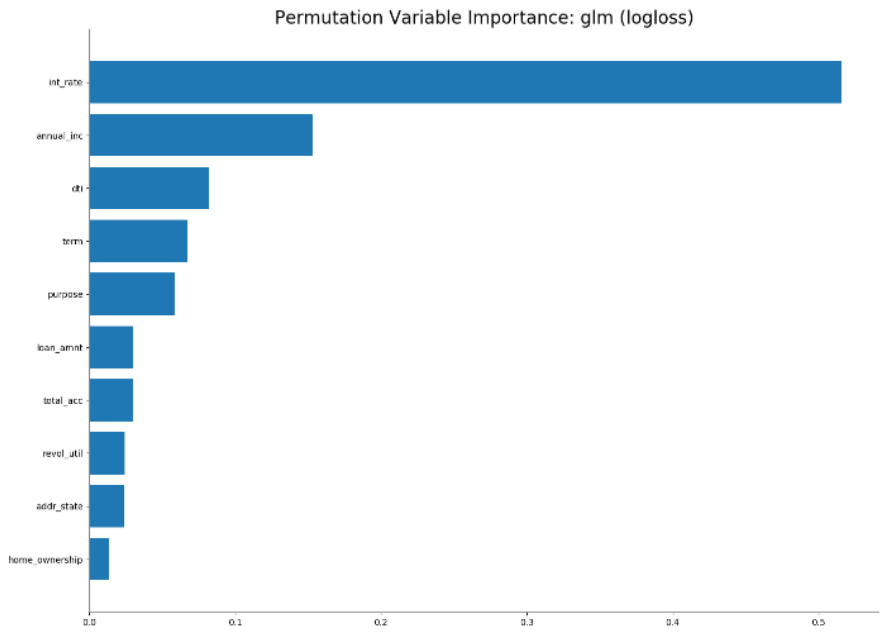
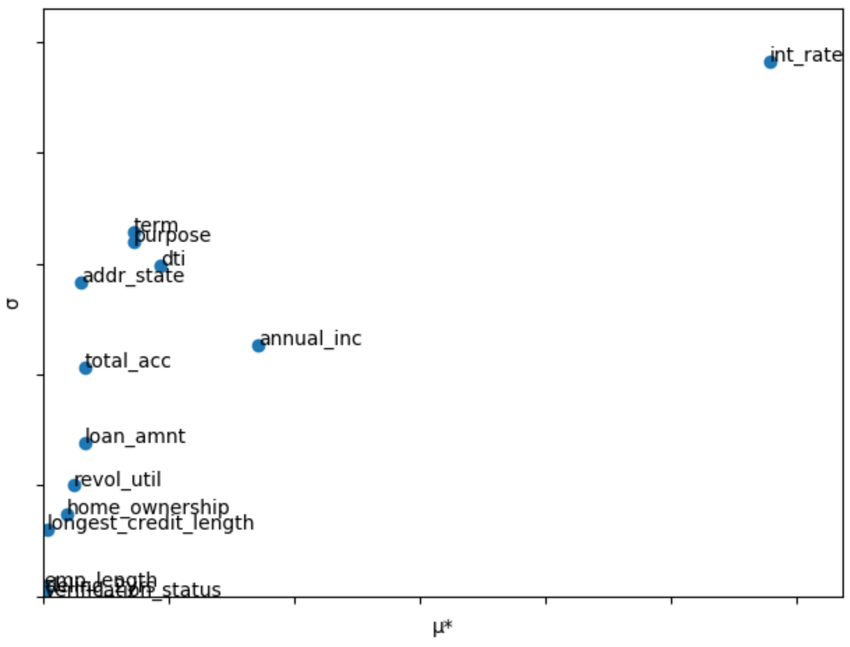


Figure 4.10: Showing OAT using Morris method with MSE



4.3 Advantages and disadvantages

The main advantage of PFI is providing global insight into the model's prediction. This technique is especially useful for non-linear models. Including every feature on the calculation while having one permuted automatically measures the interaction with the other features. Permuting features destroys the interaction effects with other features, considering both the main features effect and the interaction effects on the model performance. This comes with a disadvantage, the interaction between features is included on every iteration of the model performance measurement for every feature permuted. PFI takes as input a trained model, which means retraining the model is not required. Considering that some model's training has quite the computational cost and could be time-consuming, whereas this technique keeps the model constant and includes permuting all the features and scoring the model on every permutation. PFI has a low computational cost: the plan then requires a total number of runs that is a linear function of the number of examined features.

If two or more features are correlated unrealistic data instances can be created. Permuting one feature affects the other correlated feature, however, PFI disregards how the other might change. There can be uncertainty about whether to use training data or test data. Since PFI depends on permuting the feature, adding randomness to the measure, the results might vary one from another using OAT to repeat the permutation and averaging the measures stabilizes the measure, however, it increases the computation.

Global Sensitivity Analysis methods principle explore the entire interval of the definition of each factor. Each 'effect' for a factor is an average over the possible values of the other factors. Moreover, providing a simplified model of the input-output mapping, showing the variables of highest influence to the output, allowing the possibility to find such variables that can be disregarded upon training the model since those variables have little to no influence.

The Morris method, like all screening methods, provides sensitivity measures that tend to be capable of ranking the input factors in order of importance, but do not attempt to quantify by how much one given factor is more important than another. Therefore, combining PFI together with OAT Morris method implies a more quantitative method.

4.4 Existing Implementation of Permutation Feature Importance

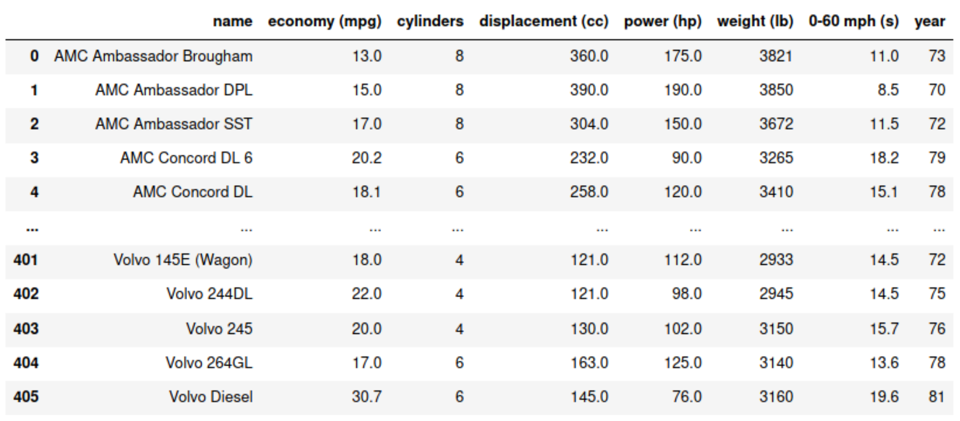
An existing PFI is from scikit learn [38] which also specifies that this method is useful for non-linear estimators. Its validation performance is measured only via R^2 or accuracy for a classifier. PFI on scikit learn can be computed either on the training set or hold-out testing or validation set. The scikit learn implementation combines both the algorithm algorithm 1 together with

Equation 3.1 in the sense that, when going through the features, in the for loop, scikit learn has a second loop repeated $k \in 1, ..., K$ times, and outside this inner loop it calculates the FI score as following:

$$FI_j = s - \frac{1}{K} \sum_{k=1}^{K} s_{k,j}$$

s denotes the loss function score, j the current feature, and k represents the number of runs. For comparing the results I shall be using the data set of cars Figure 4.11.

Figure 4.11: First rows of cars.csv dataset

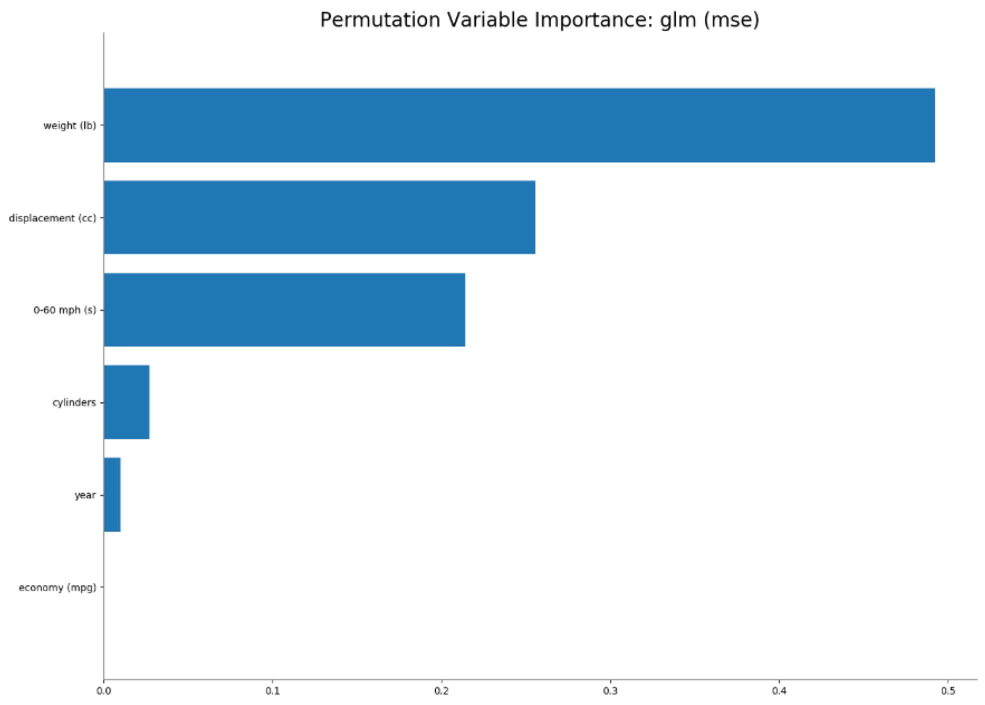


	name	economy (mpg)	cylinders	displacement (cc)	power (hp)	weight (lb)	0-60 mph (s)	year
0	AMC Ambassador Brougham	13.0	8	360.0	175.0	3821	11.0	73
1	AMC Ambassador DPL	15.0	8	390.0	190.0	3850	8.5	70
2	AMC Ambassador SST	17.0	8	304.0	150.0	3672	11.5	72
3	AMC Concord DL 6	20.2	6	232.0	90.0	3265	18.2	79
4	AMC Concord DL	18.1	6	258.0	120.0	3410	15.1	78
...	...	...	...	...	...	...	...	...
401	Volvo 145E (Wagon)	18.0	4	121.0	112.0	2933	14.5	72
402	Volvo 244DL	22.0	4	121.0	98.0	2945	14.5	75
403	Volvo 245	20.0	4	130.0	102.0	3150	15.7	76
404	Volvo 264GL	17.0	6	163.0	125.0	3140	13.6	78
405	Volvo Diesel	30.7	6	145.0	76.0	3160	19.6	81

My PFI scoring for cars dataset: weight (lb) : 0.49268, displacement (cc): 0.25558, 0-60 mph (s): 0.2143, cylinders: 0.02729, year: 0.010, economy (mpg):0.00003.

Scikit learn's PFI scoring for the same dataset: weight (lb): 0.418 (+/- 0.066), displacement (cc): 0.286 (+/- 0.048), 0-60 mph (s): 0.217 (+/- 0.044), cylinders: 0.049 (+/- 0.012).

Figure 4.12: Results of my implementation of PFI on cars dataset having "power (hp)" as a target variable



4.5 Existing Implementations of One At a Time

There is OAT Sensitivity Analysis usage for Genetic Algorithm Solving Continuous Network Design Problems [39]. The papers OAT focus is on the parameters set using genetic algorithms for continuous network design problems. Using sensitivity analysis on the effects of parameters, demonstrating that setting some parameters has clear effects on the solution. The paper uses OAT for parameter analysis and uses the 'nominal' or 'standard' value per parameter, proposing two extreme values to represent the range of likely values for each parameter. The magnitudes of the differences between the outputs are then compared to find such parameters that significantly affect the model.

Also straight from the book "Sensitivity analysis in practice: a guide to assessing scientific models" [40] which is very similar to what we've seen in the previous chapter. The model relies on a sensitivity measure, called the elementary effect, which uses incremental ratios and is a local measure. The σ and μ^* are obtained by averaging several elementary effects and their absolute values, computed at different points of the input space to lose the dependence. It attempts to explore several regions of the input space therefore the method can be regarded as global.

Chapter 5

Conclusion

This bachelor thesis provides detailed descriptions towards the importance of interpretable ML models. Also describing how more complex models labeled as black-box tend to lose interpretability to achieve better results. Elaborating on why it's crucial to trust the models output, we use Global Sensitivity Analysis to understand how the entire input affects the output. Global analysis focuses on the impact of each feature on a models numerical or other output. Random shuffling the input features OAT breaks the relationship between the feature and the target prediction. Calculating permutation feature importance values of the model indicate the dependence of the feature towards the outcome. This model-agnostic technique allows to calculate the importance of the features many times representing the average of those runs. The implementation could have been simpler by only executing PFI multiple times resulting a range of importance of each feature, however, I chose combine OAT Morris method as I believe it gives more insight to the importance of the features, and also grouping them. This method could be very time consuming in the case of big data. Having n features and m rows of the dataset (Fisher-Yates algorithm $\mathcal{O}(m)$) complexity of the permutation feature importance algorithm is $\mathcal{O}(nm)$. Having r OAT calls ($r \in [4, 10]$) calculating μ_j^* and σ_j increases the complexity to $\mathcal{O}(rn^2m) = \mathcal{O}(n^2m)$ which could have been more efficiently.

H2O's core code is written in Java, so is the implementation of Permutation Feature Importance for black-box models. There is open Github pull request of my implementation (`https://github.com/h2oai/h2o-3/pull/4610`) on H2O's library. Tested functionality of the MRTask for shuffling and PFI. MRTask testing includes testing allowed data types, Frames which have more than one Vec, and parallel runs. PFI testing includes regression and classification problems with GBM and GLM models, and testing PFI values against standardized beta values of GLM. Compared solutions with existing implementation from scikit learn. This implementation can also be used on R and Python. Tested big data files on Python and created demo on Jupyter Notebook.

When I was implementing the code it was straightforward that every feature needs to be shuffled, however, in H2O's dataset (Frame) there were such columns that aren't inherited from the input dataset. Some columns could be set by the user to be ignored, so one challenge was shuffling the Frame only on columns from the dataset, without removing the columns from the Frame. I had to learn how H2O-3 approaches storing the parsed input data and read the documentation on what data structures are being used for the input of my implementation. One thing that could be more efficient is the creation of the Frame for the PFI table. I create a H2O-3 data structure of tabular form then transform it to a another H2O-3 data structure used for parsed input data (frame), since it was impossible to pass the tabular data structure via JSON API in Python and R. While I could've implemented an efficient method to avoid those steps I decided to use existing working methods that H2O-3 already had for the table creation and permutation feature importance scores sorting. I learnt to follow existing code techniques that solved similar or subsets of the code problems I am facing.

CHAPTER 6

Glossary

black-box data goes in, decisions come out, but the processes between input and output are opaque.

Bibliography

[1] Shalev-Shwartz, S.; Ben-David, S. *Preface.* Cambridge University Press, 2014, pp. xv–xvi, doi:10.1017/CBO9781107298019.001.

[2] Foote, K. D. A Brief History of Machine Learning. *DATAVERSITY*, Mar 2019. Available from: `https://www.dataversity.net/a-brief-history-of-machine-learning/#`

[3] HEBB, D. O. *The organisation of behaviour: a neuropsychological theory*, chapter The First Stage of Perception:Growth of the Assembly. JOHN WILEY and SONS, Inc., 1952, p. 62.

[4] Biele, J. Introduction to perceptron neural network. *knoldus*, september 2017. Available from: `https://blog.knoldus.com/introduction-to-perceptron-neural-network/`

[5] Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, volume 3, no. 3, 1959: pp. 210–229.

[6] Shalev-Shwartz, S.; Ben-David, S. *Understanding Machine Learning:From Theory to Algorithms.* Cambridge University Press., 2014, 2–13 pp.

[7] Zhang, H.; Zhang, L.; et al. Overfitting and Underfitting Analysis for Deep Learning Based End-to-end Communication Systems. *Medium*, 2019, doi:10.1109/wcsp.2019.8927876.

[8] Provost, F. Glossary of Terms. *Glossary of Terms Journal of Machine Learning*, 1998. Available from: `https://ai.stanford.edu/~ronnyk/glossary.html`

[9] Soni, D. Supervised vs. Unsupervised Learning. *Medium*, Jul 2019. Available from: `https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d`

[10] Rencher, A. C.; Schaalje, G. B. *Linear models in statistics.* Wiley-Interscience, 2008, 127 – 147 pp.

[11] Kamil Dedecius, P. M. Decision Trees. 2019. Available from: https://courses.fit.cvut.cz/BIE-VZD/lectures/files/02/BI-VZD-02-en-slides.pdf

[12] Galarnyk, M. Decision Trees. *Medium*, 2019, doi:10.1002/9781119526865.ch6.

[13] Narkhede, S. Understanding AUC - ROC Curve. May 2019. Available from: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[14] Malohlava, M.; Candel, A.; et al. *Gradient Boosting machine with H2O.* H2O.ai, May 2020, 8–14 pp. Available from: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/GBMBooklet.pdf

[15] Introduction to Generalized Linear Models. Available from: https://online.stat.psu.edu/stat504/node/216/

[16] Michaelides, G. What is the difference between the general linear model GLM and generalized linear model GZLM. 05 2014.

[17] Gill, N.; Hall, P. *An introduction to Machine Learning Interpretability.* O'Reilly Media, 2018, 7–41 pp.

[18] Malgieri, G. Automated decision-making in the EU Member States: The right to explanation and other "suitable safeguards" in the national legislations. *Computer Law & Security Review*, Jul 2019. Available from: https://www.sciencedirect.com/science/article/pii/S0267364918303753

[19] Altmann, A. Permutation importance, a corrected feature importance measure. *Oxford Academic*, May 2015, doi:10.3897/bdj.4.e7720.figure2f. Available from: https://academic.oup.com/bioinformatics/article/26/10/1340/193348

[20] Robbins, B. MACHINE LEARNING: How Black is This Beautiful Black Box. Jul 2017. Available from: https://towardsdatascience.com/machine-learning-how-black-is-this-black-box-f11e4031fdf

[21] Ambiati, S. H2O Driverless AI Workshop. June 2018. Available from: https://image.slidesharecdn.com/dai-workshop-2018-print-180629201035/95/h2o-driverless-ai-workshop-69-638.jpg?cb=1530303120

[22] Ambiati, S. H2O Driverless AI Workshop. June 2018. Available from: `https://image.slidesharecdn.com/dai-workshop-2018-print-180629201035/95/h2o-driverless-ai-workshop-72-638.jpg?cb=1530303120`

[23] Breiman, L. Random forests. *Machine learning*, volume 45, no. 1, 2001: pp. 5–32.

[24] Casalicchio, G.; Molnar, C.; et al. Visualizing the Feature Importance for Black Box Models. *Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science*, 2019: p. 655–670. Available from: `https://arxiv.org/pdf/1804.06620.pdf`

[25] Ronaghan, S. classification theory decision trees and random forest. *Medium*, 2018. Available from: `https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3`

[26] christoph M. christophM/interpretable-ml-book. Available from: `https://github.com/christophM/interpretable-ml-book/blob/master/data/bike.RData`

[27] Molnar, C. *Interpretable Models*. github.io, 2019, `https://christophm.github.io/interpretable-ml-book/`.

[28] Riberio, M. T.; Singh, S.; et al. Model-Agnostic Interpretability of Machine Learning. Jun 2016. Available from: `https://arxiv.org/abs/1606.05386`

[29] Molnar, C. github.io, 2019, `https://christophm.github.io/interpretable-ml-book/`.

[30] Saltelli, A.; Ratto, M.; et al. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[31] Model calibration and validation. Available from: `https://distart119.ing.unibo.it/albertonew/?q=node/59`

[32] OLIVI, M. GLOBAL SENSITIVITY ANALYSIS. Jul 2017. Available from: `https://ec.europa.eu/jrc/en/samo/methods`

[33] Molnar, C. *Permutation Feature Importance*. github.io, 2019, `https://christophm.github.io/interpretable-ml-book/`.

[34] Fisher, A.; Rudin, C.; et al. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, volume 20, no. 177, 2019: pp. 1–81.

[35] Iooss, B.; Lemaître, P. A review on global sensitivity analysis methods. In *Uncertainty management in Simulation-Optimization of Complex Systems: Algorithms and Applications*, edited by C. Meloni; G. Dellino, Springer, 2015. Available from: `https://hal.archives-ouvertes.fr/hal-00975701`

[36] Friedman, E. AI & ML Platforms: My Fresh Look at H2O.ai Technology. Feb 2020. Available from: `https://www.h2o.ai/blog/ai-ml-platforms-my-fresh-look-at-h2o-ai-technology/`

[37] Bostock, M. Fisher–Yates Shuffle. Available from: `https://bost.ocks.org/mike/shuffle/`

[38] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

[39] Xu, M.; Yang, J.; et al. Using One-at-a-Time Sensitivity Analysis Designs for Genetic Algorithm Solving Continuous Network Design Problems. In *2009 International Joint Conference on Computational Sciences and Optimization*, volume 2, 2009, pp. 114–118.

[40] Saltelli, A. *Sensitivity analysis in practice: a guide to assessing scientific models*. Wiley, 2007, 91-110 pp.