An evaluation of Interpretation-Nets applied to Logistic Regression for explaining Neural Networks

Bachelor Thesis

presented by
Paul König
Matriculation Number 169 1336

submitted to the
Institute for Enterprise Systems
Dr. Christian Bartelt
University of Mannheim

July 2022

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 0

Abstract

Explainable AI (XAI) is widely seen as an important research topic within machine learning. In the past, multiple XAI algorithms for interpreting existing machine learning models have been proposed. These algorithms tend to have costly computation needs during the evaluation process, which make them not applicable for online interpretation settings. In addition to that, some methods do require access to the initial training data or its distribution, which can lead to privacy concerns.

Marton et al. (2022) proposed a new method for interpreting neural networks with constantly low computational effort at explanation time and no need for accessing the training data or its distribution. The $\mathcal{I}$-Net approach interprets a neural network by mapping its weights to a surrogate function, which is an intrinsically interpretable function achieving a high fidelity to the initial neural network. This approach was instantiated and evaluated for different surrogate function types, including polynomials and Decision Trees.

More research is needed to further evaluate the approach for different use cases and in particular different surrogate function types. This will lead to a holistic view of the performance of the $\mathcal{I}$-Net approach, abstracting from specific instances.

This thesis therefore introduces a new $\mathcal{I}$-Net instance with a Logistic Regression model as surrogate function. It specifies the design of the $\mathcal{I}$-Net instance, how it can be trained using synthetic data and how the data structures needed to represent the surrogate function can be implemented. Furthermore, this thesis shows empirically that this $\mathcal{I}$-Net instance can achieve competitive performance for small feature sets in comparison to $\mathcal{I}$-Nets for Decision Trees and global surrogate modelling, i.e. a plain Logistic Regression classifier trained from the original training data.

Chapter 1

Introduction

Data modeling in general, and machine learning in particular, have a strong public presence as use cases evolve and citizens, governments, and businesses increasingly rely on the outputs of these models to make decisions [2].

Some sophisticated model types, e.g. most types of neural networks, are considered black-box models, because the learned function contains to many parameters and therefore is to complex for humans to understand [10]. In these cases, it cannot be explained which features have which influence on a prediction and how the prediction differs for different sample expressions. Explainable AI (XAI), i.e. research that uses explainable models or interprets black-box models, is therefore widely considered an important topic within artificial intelligence [2]. Most complex machine learning models have superior performance for many use cases [36], but due to social demands and in some cases regulatory requirements, especially in the financial sector [17], they require the application of separate interpretation methods to fulfill transparency requirements [2].

In the past, several methods for the interpretation of machine learning models in general and neural network specifically were proposed. These methods can be classified according to Molnar (2020) [27] across four dimensions:

- **Scope of models:**
  Model specific / model agnostic
  The specificity of the interpretation method. A interpretation method can be model-specific (i.e. limited to one concrete model type) or model-agnostic (i.e. can be applied to many different model types). This criterion is not entirely separable; some interpretation methods can be classified as neither model-specific nor model-agnostic, but fall in between these two categories.

- **Scope of explanation:**
  Local / global
  The scope of the explanation. Local interpretation methods explain a single prediction, while global interpretation methods interpret the entire model function. This criterion is also not completely separable.

- **Explanation application phase:**
  Instrinsic / post hoc
  The phase of the model-lifecycle to which the explanation method will be applied. For intrinsic explanation methods, the model will be interpreted at training time (e.g. through limiting the complexity of the model). Post hoc methods will be applied to an already trained model.

- **Type of explanation result:**
  The product provided by the explanation method. This can be categorized in four different types:

  - *Feature Summary*
    Statistics regarding the impact of single features or sets of features on the output, e.g. the feature importance.
  - *Model Internals*
    Only for model-specific approaches. Output the model-internals (i.e. learned weights) in a human-friendly way.
  - *Delta between data points*
    Only for post-hoc approaches. Query the model with many different samples and compare their outputs.
  - *Instrinsically interpretable model*
    Approximate the model with another, intrinsically interpretable model.

This thesis will focus on *global* interpretation methods, which generate an *intrinsically interpretable model*. These methods have the advantage, that they can explain the underlying function as a whole and generate an explanation result many are familiar with, which makes the interpretation widely accessible [27].

1.1 Related Work

A new XAI approach proposed by S.Marton, S.Lüdtke and C. Bartelt in 2022 is called "Interpretation Nets" [23]. Interpretation Nets ($\mathcal{I}$-Nets) can be classified as a *model specific*, *global* and *post hoc* explanation method for neural networks. To accomplish this, the $\mathcal{I}$-Net directly accesses the parameters of the neural networks (i.e. weights and biases) and maps them to a symbolic representation function (also called surrogate model), which is intrinsically interpretable and has a high fidelity with the initial neural network.

Another *global post hoc* XAI approach is called global surrogate modelling [27]. This explainable AI method should not be confused with the *surrogate model*, which is the explanation result of both the $\mathcal{I}$-Net approach and the global surrogate model approach.

Global surrogate modelling trains a separate surrogate model by using either the original training data, data drawn at random from the training data distribution or data drawn at random from another distribution. The labels are provided through querying the initial neural network itself.

$\mathcal{I}$-Nets are similar to global surrogate models, with regards to problem set and explanation result. Both methods will generate an *intrinsically interpretable model* as explanation result. One advantage of the global surrogate modelling technique over the $\mathcal{I}$-Net approach is the applicability to arbitrary model types, as this approach is *model agnostic*. $\mathcal{I}$-Nets are *model specific* and can only be applied to predetermined types of neural networks.

However, $\mathcal{I}$-Nets have multiple advantages in comparison to global surrogate modelling:

First, the global surrogate model approach requires access to the training data or to its distribution [27]. This can lead to privacy concerns. As a matter of fact, use cases with a high demand towards explainable models often have high privacy requirements as well [13].

Second, $\mathcal{I}$-Nets can be trained upfront (offline-phase). $\mathcal{I}$-Nets are functions, which are generated through a machine learning process. Each interpretation runs in constant time: For each neural network, which should be explained, exactly one query of the $\mathcal{I}$-Net is necessary [23]. Global surrogate modelling per definition does not allow any upfront offline training. For each model, which should be explained, a new machine learning model has to be trained. The runtime of this training process obviously depends on the number of samples and the hyperparameter settings. This results in a trade-off between explanation speed and explanation performance (i.e. the fidelity between the target model and the surrogate model). A sophisticated machine learning model and training process used in the global surrogate model technique may lead to high fidelity, but as well be very time consuming in the explanation phase. In contrast, the $\mathcal{I}$-Net can be a highly complex model, resulting in a high fidelity and the evaluation time will remain constant at exactly one query of the $\mathcal{I}$-Net.

However, as both methods share core concepts, they can be compared against each other. This work will focus on the $\mathcal{I}$-Net explanation approach, and compare it to global surrogate modelling as well.

$\mathcal{I}$-Nets have to be instantiated for different types of surrogate functions [23]. Instances for polynomial functions [23] and several types of decision trees [24], regarding both classification and regression problems, have been published.

1.2 Problem Statement

Evaluation results for a variety of $\mathcal{I}$-Net instance types are useful for making more general statements about the performance of this approach. This work introduces a third instance, which uses Logistic Regression as the surrogate function type and shows, that this instance can be used to explain neural networks. Logistic Regression models are easy to interpret and widely used in practise [27]. Therefore they are from interest in the context of $\mathcal{I}$-Nets. This thesis has two main contributions:

1. Proposing a specific instance of the $\mathcal{I}$-Net explanation method for classification problems based on a Logistic Regression surrogate function

2. Evaluation of the instance empirically using synthetic data in comparison to $\mathcal{I}$-Nets for Decision Trees classifiers [24] and global surrogate modelling [27], both for Logistic Regression and Decision Trees, showing that $\mathcal{I}$-Nets for Logistic Regression can achieve competitive performance for small feature sets.

Chapter 2

Theoretical Framework

The following chapter provides an overview of the $\mathcal{I}$-Net explainable AI approach for neural networks [23], the necessary steps for training an $\mathcal{I}$-Net and an specific $\mathcal{I}$-Net instance for Decision Tree *surrogate models* serving binary classification.

2.1 $\mathcal{I}$-Nets as a global interpretation method for neural networks

$\mathcal{I}$-Nets provide explanations for existing neural networks (*post-hoc approach*), by representing a mapping function from an arbitrary neural network of a specific model family (denoted as λ-Net) to a *surrogate model g*, which has a high fidelity with the initial, more complex λ-Net, while being *intrinsically interpretable*. More formally, an $\mathcal{I}$-Net is a function

$$\mathcal{I} : \theta_\lambda \rightarrow \theta_g \ , \qquad \theta_\lambda \in \Theta_\lambda \ , \ \theta_g \in \Theta_g$$

where λ and g are functions of the kind

$$\begin{array}{rll} \text{For Classification:} & \lambda : X \rightarrow P(Y|X) & g : X \rightarrow P(Y|X) \\ \text{For Regression:} & \lambda : X \rightarrow Y & g : X \rightarrow Y \end{array}$$

where the set X is called feature-set and the set Y is called regression- or classification-target-set.
Throughout this thesis I will use the following conventions regarding terminology:

λ-Net	the initial neural network, which should be explained
$\mathcal{I}$-Net	the neural network used to fulfil the interpretation task
surrogate model g	the symbolic representation created by the $\mathcal{I}$-Net

As the function $\mathcal{I}$ relies directly on the parameters of the λ-Net, the λ-Net function has to be represented in a standard way (i.e. a consistent data model for the model parameters has to be established). Therefore, Θ_λ and Θ_g, the sets of representations of all possible functions λ and g, were introduced.

$\mathcal{I}$-Nets itself are implemented as neural networks, i.e. the function $\mathcal{I}$ is derived through a machine learning task. As a matter of fact, the approach was presented by Marton et al. with the slogan "Explanations for neural networks by neural networks" [23]. The training process of the $\mathcal{I}$-Net neural network can be generalized by three steps [23]:

1. Synthetic Data Generation
   Creation of synthetic data (X, Y) for the training of the λ networks

2. λ-Net training
   Training a set of neural networks (λ-Nets) and extraction of their weights

3. $\mathcal{I}$-Net training
   Training one neural network ($\mathcal{I}$-Net) with the extracted weights as samples

Figure 2.1 gives an overview of the training process, the needed data structures and the main steps involved in training an $\mathcal{I}$-Net.

The optimization metric for the $\mathcal{I}$-Net mapping function is the fidelity between λ and g. Metrics for fidelity are well-established and to some extend generic with regards to the model types of λ and g. In several $\mathcal{I}$-Net instances fidelity was computed using a distance measurement of $\lambda(x)$ and $g(x)$ [24]. This can be implemented directly in the loss function of the $\mathcal{I}$-Net.

However, a trade-off between high fidelity and good interpretability of *g* has to be considered [23]. Measurement of explainability is widely seen as an open research question and methods for measuring interpretability are not yet established [27]. Most interpretability scores provide meaningful results only for comparing functions of the same type with each other. A general interpretability score is still missing [38].

Looking at the $\mathcal{I}$-Net approach, it can be stated, that the *surrogate model* must be limited in its complexity, e.g. by choosing a simple function family and/or constraining the number of parameters.

As the $\mathcal{I}$-Net approach is *model specific*, individual instances have to be proposed. These instances differ in the surrogate function type and the λ function type, as well as in their representations Θ_g and Θ_λ.

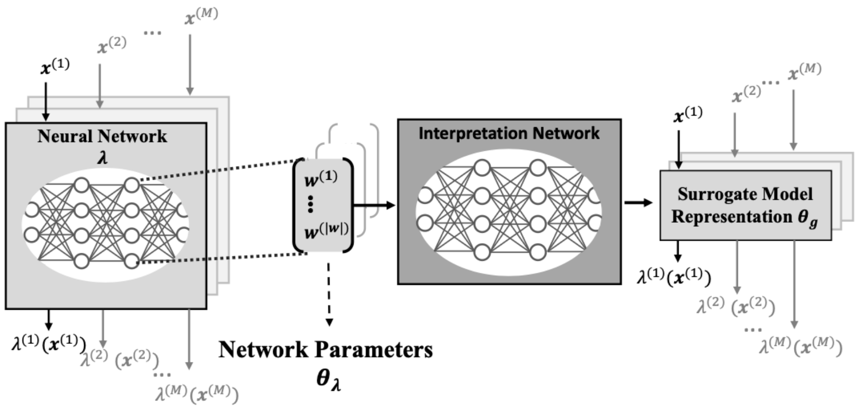


Figure 2.1: Overview of the $\mathcal{I}$-Net approach (Marton et al. [23])

2.2 $\mathcal{I}$-Nets for Decision Trees supporting binary classification

Section 2.1 covered the general definition of the $\mathcal{I}$-Net approach. This section is on a different abstraction level and summarizes a concrete $\mathcal{I}$-Net instance.

The $\mathcal{I}$-Net instance proposed by Marton et al. (2022) [24] uses a Decision Tree surrogate model and focuses on λ-Nets for classification tasks. Decision Trees are among the most commonly used algorithms for machine learning classification tasks today [31]. This is because of several advantages over other algorithms.

One advantage is the simple interpretability by humans [12] [27]. As Decision Trees can be easily visualized (see figure 2.2) and the necessary size of a Decision Tree, i.e. the number of decision nodes, with a reasonable accuracy is usually low, despite varying extensively for different datasets [35] [1], the intrinsic explainability is considered high. Therefore, the use of Decision Trees as a functional type for the symbolic representation makes sense.

Marton et al. (2022) [24] proposed $\mathcal{I}$-Net-instances for different types of Decision Trees, standard Decision Trees, Soft Decision Trees and univariate Soft Decision Trees. This thesis focuses on standard Decision Trees for classification.

Decision Trees as a classification method

Decision Trees are binary trees consisting of decision nodes and leaf nodes [3]. Each Decision node represents a split criteria, described by a feature identifier and a split value. At each decision node, the data is split into two groups: One group

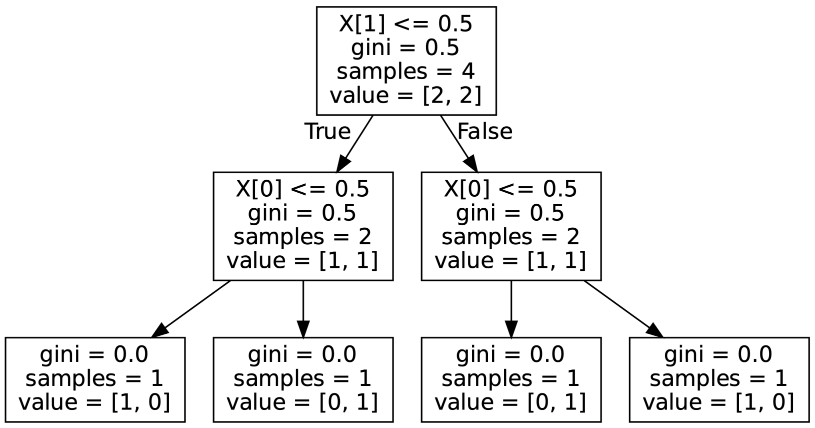


Figure 2.2: Simple Decision Tree classifier generated by scikit-learn [4] and rendered with graphviz [11]. It represents the XOR($X[0], X[1]$) function.

that satisfies the split criteria and another group that does not [3]. The feature identifier is the feature-dimension, along which the split will be performed. When the current value of the sample indicated by the dimension of the feature identifier is smaller than the split value, the sample belongs, by convention, to the left child node. In the other case, the sample belongs to the right child node [3].

All samples are pushed through the Decision Tree, from the root node (which is a decision node as well) to a leaf node. As soon as a sample has reached a leaf node, it can be classified using the majority class of the training data reaching this node. Inference is therefore simple: For every sample, the path from the root node to a leaf node, considering all splits at the decision nodes, is deterministic [3].

The length of the longest path from the root node to a leaf node is denoted as the maximum depth of the Decision Tree. Figure 2.2 shows a simple Decision Tree with a depth of 2.

The training of a Decision Tree is done using a loss functions (also denoted as split criteria) which rates the individual splits. One example of a loss function is the gini impurity index [3]. The gini impurity index in general is a measure for the similarity of elements in a set of data points (the more similar, the less *impure* and the lower the gini impurity index). For the case of binary classification, the gini impurity index for a set S of datapoints is defined as [3]

$$gini(S) = 2 \cdot f_0^S \cdot f_1^S$$

$$\text{where} \quad f_0^S = \text{fraction of elements in } S \text{ assigned to class false}$$
$$f_1^S = \text{fraction of elements in } S \text{ assigned to class true}$$

For application in the loss function of Decision Trees, the gini impurity index has to be computed for two sets of data points. The set S_l represents all the data points that fall into the left child node (i.e. all samples which meet the split criteria) and the set S_r represents the data points in the right child node (i.e. all samples which not meet the split criteria). In the *weighted gini* index, the gini index for both datasets is calculated and then weighted with the number of samples in each set. More formally, the $weightedgini$ impurity index is calculated as [3]

$$weightedgini(S_l, S_r) = \frac{|S_l|}{|S_l| + |S_r|} \cdot (2 \cdot f_0^{S_l} \cdot f_1^{S_l}) + \frac{|S_r|}{|S_l| + |S_r|} \cdot (2 \cdot f_0^{S_r} \cdot f_1^{S_r})$$

$$\text{where} \quad f_0^S = \text{fraction of elements in } S \text{ assigned to class false}$$
$$f_1^S = \text{fraction of elements in } S \text{ assigned to class true}$$

The goal of a generic Decicision Tree optimizer is to minimize the *weightedgini* for each decision node. $\mathcal{I}$-Nets are using a different loss function, as the training setting is different.

$\mathcal{I}$-Net instance for Decision Trees

The definition of the $\mathcal{I}$-Net loss function is to be made for each instance independently. As described in section 2.1, most $\mathcal{I}$-Net instances use a distance measurement between $\lambda(x)$ and $g(x)$. For the $\mathcal{I}$-Net for Decision Trees instance, the loss function was represented through a mean binary cross-entropy. For each λ-Net in a set of λ-Nets, the binary cross entropy from θ_λ and θ_g, where $\theta_g = \mathcal{I}(\theta_\lambda)$ is calculated and then the average is drawn. More formally, the loss $\mathcal{L}_\mathcal{I}$ is computed as

$$BC(\theta_\lambda, \theta_g) = \frac{1}{M} \sum_{j=1}^{M} \lfloor \lambda(x^{(j)}) \rceil \cdot \log(g(x^{(j)})) \\ + (1 - \lfloor \lambda(x^{(j)}) \rceil) \cdot \log(1 - \log(g(x^{(j)})))$$

$$\mathcal{L}_{\mathcal{I}} = \frac{1}{|\Theta_\lambda|} \sum_{\theta_\lambda \in \Theta_\lambda} BC(\theta_\lambda, I(\theta_\lambda))$$
$$\text{where} \quad \lfloor \cdot \rceil = \text{round to the nearest integer}$$

as proposed by [24].

Using this loss function, the optimization goal of a high fidelity between λ and g can be accomplished. As described in 2.1, the trade-off between fidelity and interpretability of g has to be considered as well. In the case of Decision Tree classifiers, the interpretability depends mainly on the number of decision nodes (see above). The number of decision nodes can be constrained using the max depth hyperparameter.

The number of total nodes of a fully grown binary tree according to [25] is

$$2^{d+1} - 1$$
$$\text{where} \quad d = \text{max depth}$$

Specifically for the $\mathcal{I}$-Net approach, the Decision Trees representing g are always fully grown [24]. A fully grown binary tree contains exactly 2^d leaf nodes [25]. Hence, all other nodes, $2^d - 1$, are decision nodes (counting the root node as decision node). Therefore, the max depth hyperparameter should generally be kept small, to keep the number of decision nodes small. However, it should be large enough, so that the performance, i.e. the fidelity of λ and g, does not suffer [24].

Besides from defining a loss function and ensuring the interpretability of the resulting *surrogate function* g, suitable representations Θ_g for g and Θ_λ for the λ-Net have to be proposed.

The representation of the λ-Net neural network parameters Θ_λ was proposed in [24] as a flat list of all weights and biases in a layer-wise order.

The Decision Tree classifier representation for the *surrogate function* g was defined as a flat list containing three parts for the feature identifiers, split values and class probabilities for leaf nodes. More formally, Θ_g is defined as [24]:

$$\Theta_g = \mathcal{F}_1\mathcal{F}_2 \ldots \mathcal{F}_n\mathcal{V}_1\mathcal{V}_2 \ldots \mathcal{V}_n\mathcal{P}_1\mathcal{P}_2 \ldots \mathcal{P}_m$$
$$\begin{aligned} \text{where} \quad n &= \text{number of decision nodes} \\ m &= \text{number of leaf nodes} \\ \mathcal{F}_i &= \text{Feature identifier in decision node } i \\ \mathcal{V}_j &= \text{Split value in decision node } j \\ \mathcal{P}_k &= \text{Class propability in leaf node } k \end{aligned}$$

In this case, the order of the decision nodes is defined as a breadth-first search [5]. The order of the leaf nodes is from left to right.

The training of the $\mathcal{I}$-Net for Decision Trees is following the already known three steps of Synthetic Data Generation, λ-Net training and $\mathcal{I}$-Net training. For the Synthetic Data Generation, a new data generation method was used, which draws samples from multiple distributions with random parameter settings, to ensure application for many real-world use cases [24].

The $\mathcal{I}$-Net instance for Decision Trees was evaluated by Marton et al. (2022) [24] and achieved a significantly higher fidelity than the comparison method, which was symbolic regression [26], given no access to the training data or its distribution. This thesis compares a new instance, $\mathcal{I}$-Nets for Logistic Regression, with the existing instance for Decision Trees and against global surrogate modelling (as described in 1.1), both for Logistic Regression and Decision Trees.

Chapter 3

$\mathcal{I}$-Nets for Logistic Regression

Logistic Regression is one of the most widely used machine learning algorithms for classification [37]. Therefore, the investigation, whether $\mathcal{I}$-Nets with a Logistic Regression model as surrogate representation function can achieve competitive performance regarding fidelity, is from high interest.

In this chapter I summarize the main characteristics of Logistic Regression as a machine learning algorithm, propose an $\mathcal{I}$-Net-Instance for Logistic Regression and introduce a suitable training method for it.

3.1 Logistic Regression as a universal method for data modelling

Logistic Regression is (like Decision Trees as well) a *discriminative* classification algorithm, i.e. it learns directly a decision boundary in the feature space to separate the different classes from each other. This is in contrast to *generative* classification algorithms, which model the complete feature data distribution for each class [20].

Most *discriminative* classifications algorithms learn a function, which directly maps a sample to the different probabilities for each class [20]. In the case of binary logistic regression, only two possible classes can occur, either true (1) or false (0). The problem set for a binary Logistic Regression is therefore to learn a mapping function f from an independent feature vector X to a dependent class propability $Y = 1$, where X is a vector of length n and $Y \in [0, 1]$. More formally, f is defined as [20]

$$f : X \to P(Y = 1 \,|\, X)$$

The propability for $Y = 0$ can be computed through

$$P(Y = 0 \,|\, X) = 1 - P(Y = 1 \,|\, X)$$

Logistic Regression and Linear Regression are related, as the Logistic Regression model is based on a linear model and the logistic function σ, also called sigmoid function. The additional sigmoid function is necessary to force the output of the model to a value between 0 and 1 [20].

The sigmoid function σ (also known as logistic function) is defined as [20]

$$\sigma(y) = \frac{1}{1 + \exp(y)}$$

The linear model can be generalized as [20]

$$y = b + \sum_{i=1}^{n}(w_i \cdot x_i) \qquad \text{, where } x_i \in X$$

Often, b is called the bias parameter and $w_i \in W$ are called the weight parameters. The bias exists exactly once and the weights $w_i \in W$ exist once for each feature $x_i \in X$, i.e. n-times.

Combining both, sigmoid function and linear model together, the Logistic Regression model is defined as

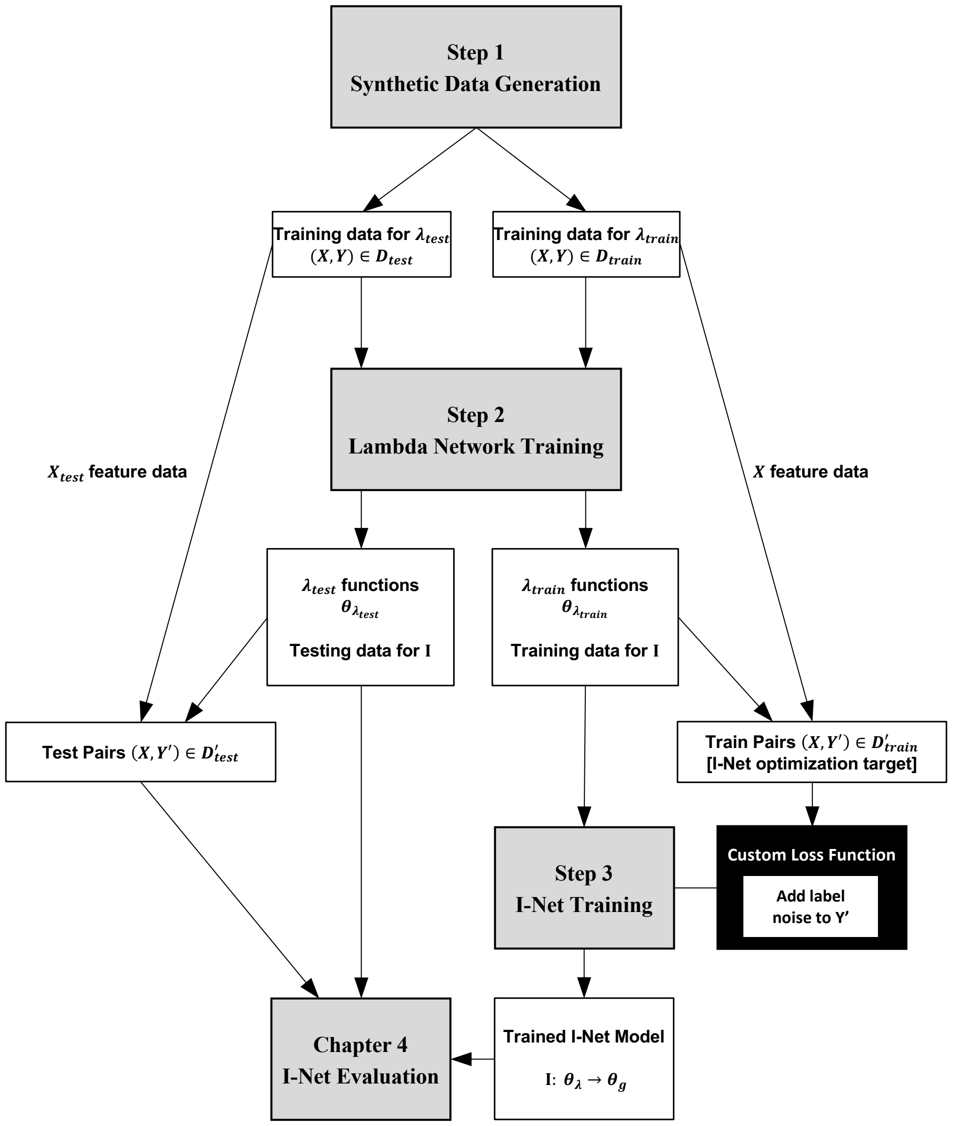
$$\begin{aligned} P(Y = 1 \,|\, X) = f(X) &= \frac{1}{1 - \exp(y)} \\ \text{where} \qquad y &= b + \sum_{i=1}^{n}(w_i \cdot x_i) \\ \text{where} \qquad x_i &\in X \end{aligned}$$

In many cases, the training of a Logistic Regression model is done through maximum likelihood estimation in combination with a Binary Cross-Entropy loss function.

In the training process for the model parameters b and $w_i \in W$ for $i = 1, \cdots, n$ with m training samples (X, Y), Binary Cross-Entropy Loss is defined as [28]

$$loss(X, Y) = -\frac{1}{m}\sum_{j=1}^{m}(y_j \cdot \log(f(x_j)) + (1 - y_j) \cdot \log(1 - f(x_j)))$$

3.2 $\mathcal{I}$-Net instance for Logistic Regression



Step 1
Synthetic Data Generation
Training data for λ_{test}
$(X, Y) \in D_{test}$
Training data for λ_{train}
$(X, Y) \in D_{train}$
Step 2
Lambda Network Training
X_{test} feature data
X feature data
λ_{test} functions
$\theta_{\lambda_{test}}$
Testing data for I
λ_{train} functions
$\theta_{\lambda_{train}}$
Training data for I
Test Pairs $(X, Y') \in D'_{test}$
Train Pairs $(X, Y') \in D'_{train}$
[I-Net optimization target]
Step 3
I-Net Training
Custom Loss Function
Add label noise to Y'
Chapter 4
I-Net Evaluation
Trained I-Net Model
I: $\theta_\lambda \rightarrow \theta_g$

Figure 3.1: Overview of the $\mathcal{I}$-Net instance training process

The goal of the $\mathcal{I}$-Net instance for Logistic Regression is to interpret λ-Nets classification models through a Logistic Regression surrogate function g. The process of training the $\mathcal{I}$-Net can be divided into three steps, Synthetic Data Generation,

λ-Net Training and $\mathcal{I}$-Net Training. Figure 3.1 shows the different steps and their interfaces, as well as the different data structures.

Step 1: Synthetic Data Generation

A set $\mathcal{D}$ of n datasets (X, Y), each containing m samples (x, y), is created through the make classification algorithm [29] [14]. The samples are generated through a normal distribution and each class is represented by the same number of samples. As this instance serves binary classification, the number of classes $|y|$ is fixed to two, while the number of features $|x|$ varies. In the data generation process the feature vector is split up into informative, redundant, duplicate and useless features [29] [14]. Informative features are the only type of features which provide useful new information to the classifier. Redundant features are linear combinations of the informative features and therefore do not provide additional information. Duplicate features are repeated features, either of the informative or the redundant features. Useless features contain random noise.

The numbers of total features t, informative features i, redundant features r, duplicate features d and useless features u are hyperparameters, where:

$$i + r + d + u = t$$

To ensure, that the $\mathcal{I}$-Net generalizes well for different λ-Nets, the numbers of informative, redundant and duplicate features are set at random and independently for each dataset $(X, Y) \in \mathcal{D}$. The total number of features t is the same for each dataset. The number of useless features is fixed at 0, because any noise introduced by the useless features will be abstracted by the λ-Net and therefore does not impact the $\mathcal{I}$-Net. Noise is added in the $\mathcal{I}$-Net loss function instead (see step 3). The random parameter setting introduces another hyperparameter s, $0 <= s <= t$, which controls the range for the random number of informative features. Given t and s, the different hyperparameters are set as follows:

$$\begin{aligned} i &= \text{randomInt}(t - s, t) \\ r &= \text{randomInt}(0, t - i) \\ d &= t - i - r \\ u &= 0 \end{aligned}$$

$$\text{where} \quad \text{randomInt}(a, b) = \text{random integer } x \text{ with } a <= x <= b$$

The data generation process is formalized in algorithm 1.

Algorithm 1

```
DataGeneration($n$, $m$, $t$, $s$)
 1: $\mathcal{D} \leftarrow []$
 2: for j = 0, ..., n-1 do
 3:     $i \leftarrow$ RandomInt($t - s$, $t$)
 4:     $r \leftarrow$ RandomInt(0, $t - i$)
 5:     $d \leftarrow t - i - r$
 6:     $u \leftarrow 0$
 7:     $(X, Y) \leftarrow$ MakeClassification($m$, $i$, $r$, $d$, $u$) ▷ [29] [14]
 8:     $\mathcal{D} \leftarrow \mathcal{D} \cup (X, Y)$
 9: end for
10: return $\mathcal{D}$
```

Step 2: Lambda Network Training

For each of the n datasets generated through the synthetic data generation a λ-Net neural network is trained. This is done using the m samples contained in the dataset. The parameters of all trained λ-Nets (n different instances) are then stored using a representation Θ_λ.

The λ-Net parameters consist of weights and biases for each layer of the neural network. In the representation Θ_λ, the parameters are stored as a one-dimensional array, following a sequential order by the layers

$$\Theta_\lambda = w_0\, b_0\, w_1\, b_1 \cdots w_n\, b_n$$

$$\begin{aligned} \text{where} \quad n &= \text{number of layers in the neural network} \\ w_i &= \text{sequence of weights at layer } i \\ b_i &= \text{sequence of biases at layer } i \end{aligned}$$

This representation could be improved regarding the level of contained information and applicability for different λ-Netlayouts. These possible improvements are described in section 5.3.

After training a λ-Net, the function λ is used to predict $Y' = \lambda(X)$, resulting in a set $\mathcal{D}'$ of datasets (X, Y'). The predictions Y' differ from Y, because the λ-Net may not have modelled the initial dataset (X, Y) perfectly. As the $\mathcal{I}$-Net is supposed to learn the function λ and not the function used by the synthetic data generation method, the $\mathcal{I}$-Net training and evaluation process uses $\mathcal{D}'$ instead of $\mathcal{D}$. Using $\mathcal{D}$ would introduce a specific weak supervision problem [16] into the training process, i.e. the labels would not represent the actual learning task properly.

Algorithm 2 formalizes the λ-Net training routine.

Algorithm 2

```
λ-NET-TRAINING($\mathcal{D}$)
 1: $\theta_\Lambda \leftarrow []$
 2: $\mathcal{D}' \leftarrow []$
 3: for all $(X, Y) \in \mathcal{D}$ do
 4:     $\lambda \leftarrow$ TRAINLAMBDANET$(X, Y)$
 5:     $Y' \leftarrow \lambda(X)$
 6:     $\mathcal{D}' \leftarrow \mathcal{D}' \cup (X, Y')$
 7:     $\theta_\lambda \leftarrow$ GETFUNCTIONREPRESENTATION$(\lambda)$
 8:     $\theta_\Lambda \leftarrow \theta_\Lambda \cup \theta_\lambda$
 9: end for
10: return $\theta_\Lambda, \mathcal{D}'$
```

The resulting sets $\mathcal{D}'$ and θ_Λ are partitioned into training sets $\mathcal{D}'_{train}$ and $\theta_{\Lambda_{train}}$, used in the $\mathcal{I}$-Net loss function, and testing sets $\mathcal{D}'_{test}$ and $\theta_{\Lambda_{test}}$, used in the $\mathcal{I}$-Net evaluation.

Step 3: $\mathcal{I}$-Net Training

As described in chapter 2, $\mathcal{I}$-Nets are functions $\mathcal{I} : \theta_\lambda \rightarrow \theta_g$, where θ_λ is the parameter representation of a λ-Net and θ_g is the parameter representation of a surrogate function.

In order to train and apply an $\mathcal{I}$-Net, the representation Θ_g has to be defined for g being a Logistic Regression model. Logistic Regression parameters contain exactly one bias and n weights, where n is the number of features. The representation Θ_g stores the parameters as a one-dimensional array containing the bias followed by the weights.

$$\Theta_g = b\, w_0\, w_1\, \cdots\, w_n$$

$$\begin{aligned} \text{where} \quad n &= \text{number of features} \\ b &= \text{bias} \\ w_i &= \text{weight for feature i} \end{aligned}$$

The Interpretation Network, despite representing a function $\mathcal{I} : \theta_\lambda \rightarrow \theta_g$, will be trained using pairs of $(X, Y') \in \mathcal{D}'_{train}$ and the corresponding function representations $\theta_\lambda \in \theta_{\Lambda_{train}}$ with $\lambda(X) = Y'$. To accomplish the optimization goal, to maximize the fidelity of λ and $g = \mathcal{I}(\lambda)$ for arbitrary λ-Nets, a custom loss function is needed.

The custom loss function receives training data $(X, Y') \in \mathcal{D}'_{train}$ and the predicted surrogate function representation $\theta_g = \mathcal{I}$-Net(θ_λ) (for the current λ-Net). For each pair, the loss function queries the surrogate model g with the training data X and computes the Binary Cross Entropy from the resulting predictions $\hat{Y} = \lambda(X)$ and Y'. The Binary Cross Entropy Loss, also known as log loss, is discussed in section 3.1. Binary Cross Entropy Loss is suited for this problem set, as it is widely used and offers fast convergence [22].

To ensure that a model has learned the function $\mathcal{I}$ for unknown λ-Nets and does not overfit on the training data, random noise can be added through the custom loss function. The noise is applied as label noise to the predictions Y'. The noise is added only during the training process, i.e. only to training pairs $(X, Y') \in \mathcal{D}'_{train}$.

This type of noise can be used to validate, whether a specific model instance is overfitting on the training data. If the performance for the same model drops significantly, as soon as training noise is added, this is an indicator for overfitting [7]. In this case, the model learned not only the desired function, but also the noise on the training data, which leads to worse performance on the testing data (which does not have noise applied to).

The amount of noise is controlled trough the hyperparameter $noise$ (if set to 0, no noise is added) and is applied as label noise to the predictions Y' of each dataset. A random fraction of size $noise$ from predictions Y' are flipped. The flip of a single sample $y' \in [0, 1]$ is done by

$$\overline{y'} = 1 - y'$$

Algorithm 3 presents a formal notation of the custom loss function. Most of the time, this loss function will be queried in batches. The custom loss functions then returns a list of the computed losses for all training samples in the batch.

Algorithm 3

```
CUSTOM_LOSS($(X, Y')$, $\theta_g$, $noise$)
1: if $noise > 0$ then
2:     $Y' \leftarrow$ FLIPFRACTION($Y'$, $noise$)
3: end if
4: y_true $\leftarrow Y'$
5: y_pred $\leftarrow g(X)$
6: loss $\leftarrow$ BINARYCROSSENTROPY(y_true, y_pred)
7: return loss
```

Chapter 4

Evaluation

In this chapter, I show that $\mathcal{I}$-Nets for Logistic Regression can be implemented as described in chapter 3 and offer suitable performance for the practical application as an explanation method for neural networks. The performance of the $\mathcal{I}$-Net will be evaluated along the fidelity between λ and g in comparison to $\mathcal{I}$-Nets for Decision Trees [24] as described in 2.2, plain Logistic Regression and plain Decision Trees (through the global surrogate model technique [27]).

4.1 Evaluation and comparison metrics

For the evaluation of the fidelity and the comparison of the different models, the following metrics will be used. Please note that the appendix C provides more standard metrics for reference.

Fidelity score

The fidelity score is an adaption of the F_1 metric. The F_1 metric is used instead of the accuracy score, as it is more robust to imbalanced datasets [18]. Although imbalanced datasets are probably rare in this setting (as the initial data generation method generates balanced datasets), they can occur through imperfections in the λ-Net training process. Therefore, the F_1 score is a better choice.

The F_1 score is a metric for the prediction accuracy of a classifier. It is the harmonic mean value of two other metrics, the precision and the recall (for binary classifiers sometimes called sensitivity). Both of them rely on the four primary statistical numbers for classification accuracy, the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). The precision measures, how well the model predicts correctly that a sample is positive in relation to how well it prevents false positives [18]:

$$precision = \frac{TP}{TP + FP}$$

The recall measures, how well the model predicts a positive sample correctly in relation to how well it prevents false negatives [18]:

$$recall = \frac{TP}{TP + FN}$$

The F1 score is defined as the harmonic mean of both values [18]:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The fidelity score is computed as the mean of the F_1 metric for each testing dataset in $\mathcal{D}'_{test}$. The exact computation of the fidelity score for the $\mathcal{I}$-Net for Logistic Regression model is presented in algorithm 4. Computation for the other models is done similar with respect to the specific data representation. In all cases, the score is calculated using a test split of the originally generated set of datasets.

Algorithm 4

```
FIDELITY SCORE($\mathcal{I}$, $\theta_{\Lambda_{test}}$, $\mathcal{D}'_{test}$)
 1: scores ← []
 2: for all $\theta_\lambda \in \theta_{\Lambda_{test}}$ do
 3:     tn, fp, fn, tp ← 0, 0, 0, 0
 4:     $\theta_g \leftarrow \mathcal{I}(\theta_\lambda)$
 5:     $X_{test}$ ←GATHER TEST SAMPLES FOR($\lambda$)
 6:     for all $x_{test} \in X_{test}$ do
 7:         $y \leftarrow \lambda(x_{test})$
 8:         $\hat{y} \leftarrow g(x_{test})$
 9:         tn, fp, fn, tp ←UPDATECONFUSIONMATRIX(tn, fp, fn, tp, $y$, $\hat{y}$)
10:     end for
11:     scores ← scores ∪ $F_1$(tn, fp, fn, tp)
12: end for
13: return MEAN(scores)
```

Receiver Operating Characteristic (ROC)

In addition to the fidelity score, ROC-AUC scores are used to provide a measure of the degree of separation the surrogate function g achieves between the two classes. The Receiver Operating Characteristic (ROC) [33] is a well known evaluation method for binary classifiers. It assumes that a good classifier will elaborate a clear separation by the feature data between the classes, and a bad classifier will suffer to split the classes apart from each other. The ROC is often visualized as a two-dimensional plot, composed of an x-axis for the true-positive rate and an y-axis for the false-positive rate. Both are bound to values from $[0, 1]$. This illustrates the trade-off between setting a high threshold and a low threshold: A low threshold will lead to a high true-positive rate, but will likely lead to a high false-positive rate as well. The corresponding threshold itself cannot be determined directly from the classic ROC curve visualisation.

The figure 4.1 shows two different ROC curves α and β and a dashed diagonal line. The dashed diagonal line represents a random guess. For each increase in true-positive rate, the false-positive rate will increase by the same amount. Looking at α and β, both of them are achieving a better separation of the classes than a random guess, however, β is intuitively better, as it offers a higher true-positive rate for the same false-positive rate. This intuitive understanding can be quantified with the AUC score, the area underneath the curve. The AUC score is defined as the ratio of the surface area between the x-axis and the ROC curve in the whole square area of size 1 x 1. A perfect algorithm will have a ROC-AUC score of 1, as the predictions are perfectly correlated with the ground truth. A score of 0.5 indicates that there is no correlation of the predictions and the true values, while a score of 0 indicates, that predicted values and true values are perfectly negatively correlated. In this example, α has a ROC-AUC score of 0.63 and β has a ROC-AUC score of 0.76.

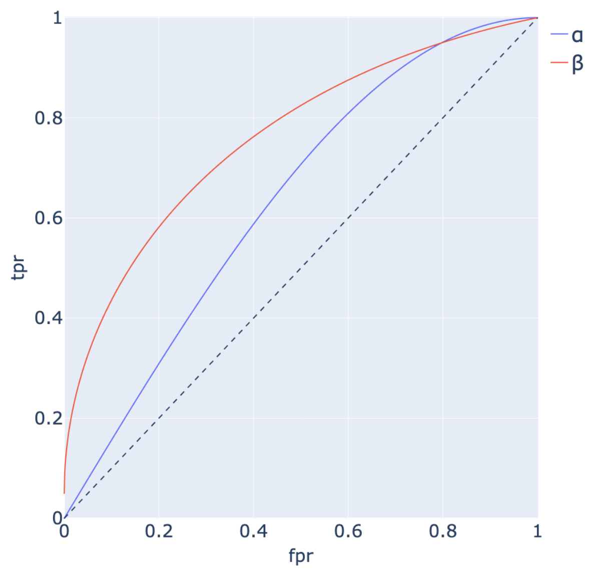


Figure 4.1: Example ROC curve

Supporting the evaluation, the plotted ROC curves and calculated ROC-AUC scores are determined using all test data at the same time, i.e. for $\bigcup_{(X,Y')\in\mathcal{D}'_{test}}(X,Y')$.

Taking the interpretability into account

While the evaluation itself focuses mainly on the fidelity of the approaches, as this is unambiguously measurable, the interpretability of the surrogate function is from high interest as well. As described in 2.1, high fidelity and high interpretability can not be optimized independently.

It can be argued, that the interpretability of a model can be quantified with the Interpretability Score [12], which can be formulated for a Logistic Regression classifier as the the number of parameters the model contains (see section 2.1). Despite the lack of real comparability between different function types (see section 2.1 as well), it makes sense to compare the number of weights of a Logistic Regression model to the number of decision nodes contained in a Decision Tree. Both, decision nodes for trees and weights for logistic regression, are linked to a feature (through the feature identifier, or the weight index respectively) and contain exactly one arbitrary numeric value (the split value, or the weight value respectively). Therefore, it can be stated, that having similar numbers of decision nodes and Logistic Regression parameters, makes both models comparable regarding their interpretability.

As the number of coefficients in the Logistic Regression classifier is determined by the number of features, it is fixed for a specific instance and as a consequence, interpretability is fixed for a specific experiment. Therefore, the number of decision nodes in the Decision Trees have to be controlled and set to approximately the number of parameters in the Logistic Regression classifier.

4.2 Experimental Setup

All scores are determined using a testing set Λ_{test} of size 1'000, where each $\lambda_{test} \in \Lambda_{test}$ contains 5'000 samples.

$\mathcal{I}$-Nets for Logistic Regression

In this evaluation, three different hyperparameter settings for $\mathcal{I}$-Nets for Logistic Regression are evaluated. For each hyperparamter configuration, 10'000 datasets with 5'000 samples each were generated. The configurations differ in the total number of features $\{5, 10, 20\}$. Thereby, the behavior of the $\mathcal{I}$-Net for small feature sets and large feature sets can be evaluated. All configurations make use of the three different kinds of features: informative, redundant and repeated. The hyperparameter s for setting the different types of features at random is set, with regards to the number of total features n, to {n=5: 2, n=10: 5, n=20: 10}, i.e. at least half of the features (for n=5 slightly more) are guarantied to be informative.

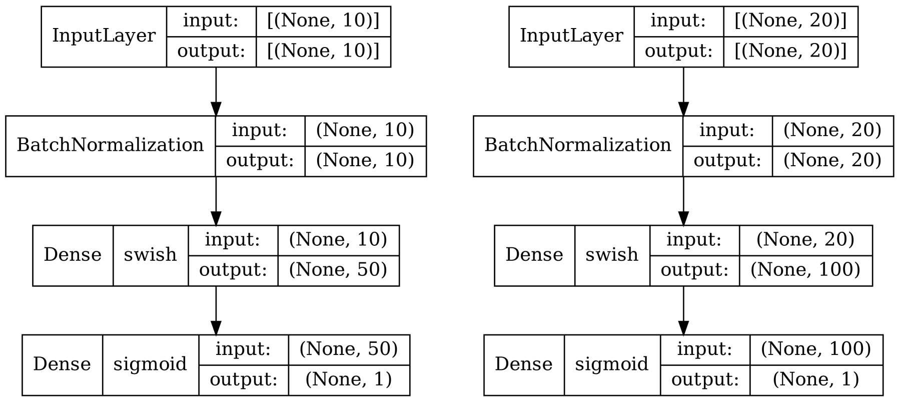
For a more detailed description of the algorithm for the synthetic data generation, please refer to 3.2. The algorithm was implemented using the scikit-learn make_classification API [4].

For each generated dataset, one λ-Net is trained using all the 5'000 samples. The neural network implementation of the λ-Nets is supported by the keras Sequential API [9]. As the number of features is small and sophisticated performance of the λ-Nets is not the main optimization criteria of this experimental setting, the λ-Net architecture is a simple feed-forward neural network with one hidden fully connected layer consisting of 50 neurons (for n=5 and n=10) and 100 neurons (for n=20). This takes the larger inputsize into account.

The input gets compressed using batch normalization, as this is widely considered a best practise [32].

The λ-Net architecture is shown in figure 4.2.

The 10'000 trained λ-Net samples are split in a train and test split for the $\mathcal{I}$-Net itself. This experiment uses 9'000 samples for training and 1'000 for testing.



(a) n = 10 (b) n = 20

Figure 4.2: λ-Net architectures for $n = 10$ and $n = 20$, the architecture for $n = 5$ is equal to that from $n = 10$, except for the shape of the input layer

The $\mathcal{I}$-Net follows a traditional deep feed-forward neural network architecture, consisting of three hidden layers with 2048, 1024 and again 512 neurons. After each hidden layer, a dropout of 0.5 is added. The $\mathcal{I}$-Net uses the swish activation function in the hidden layer. Swish is used instead of Rectified Linear Unit (ReLU), because experiments have shown that it constantly achieves the same or better performance than ReLU [30]. Swish is defined as a combination of a linear activation function and sigmoid:

$$swish(x) = x \cdot sigmoid(x)$$

The architecture is visualized in figure 4.3.

The $\mathcal{I}$-Net is optimized through the Adam optimizer. It offers fast convergence in general [21] and tends to be a good choice for other $\mathcal{I}$-Net instances [24] [23]. The loss function is application specific and was described in section 3.2. The $\mathcal{I}$-Net for Logistic Regression does not use Batch Normalization, as prior experiments have shown slightly higher performance without it.

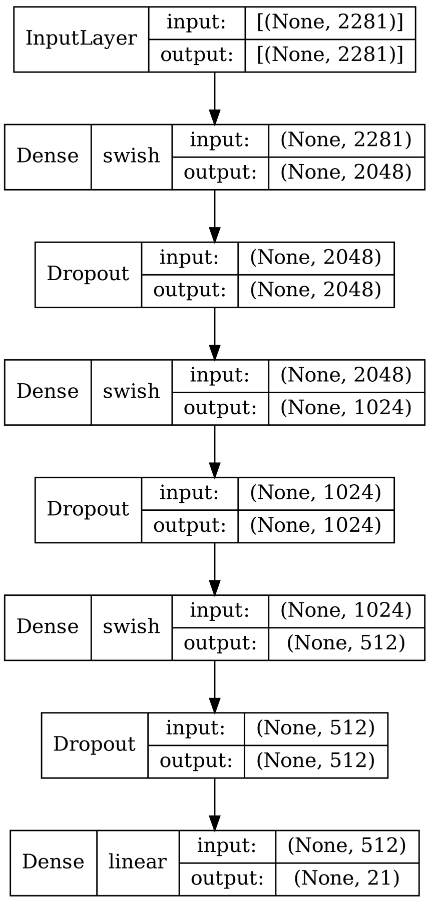


Figure 4.3: Interpretation Network architecture for Logistic Regression

For a full list of all hyperparameters please refer to appendix B.

Plain Logistic Regression

The proposed $\mathcal{I}$-Net instance for Logistic Regression will be compared to the global surrogate model technique (section 1.1 [27]), i.e. a traditional Logistic Regression model. It is trained using parts of the $\mathcal{I}$-Net for Logistic Regression training data, the original feature data X and the predictions $Y' = \lambda(X)$. More formally, the classifier will be trained and evaluated for each pair $(X, Y') \in \mathcal{D}'_{test}$ independently. Each dataset (X, Y') itself is split up into 4250 training samples and 750 test samples. The implementation is supported by the scikit-learn LogisticRegression API [4].

$\mathcal{I}$-Nets for Decision Trees

For experiments with the $\mathcal{I}$-Net instance for Decision Trees, the implementation from Marton et al. [24] is used. For a detailed overview of implementation details, please refer to that paper.

The set of hyperparameters for the $\mathcal{I}$-Net approach in general can be split up into four different parts:

1. Parameters controlling Synthetic Data Generation, e.g. number of samples
2. Network parameters for the λ-Net
3. Network parameters for the $\mathcal{I}$-Net
4. Function family and contraints for g, e.g. max depth for Decision Trees

The four different subsets of hyperparamters were optimized differently.

The parameters controlling the generation of synthetic data and the λ-Net were chosen to be the same for both the $\mathcal{I}$-Net for Logistic Regression and the $\mathcal{I}$-Net for Decision Trees. This ensures a controlled and comparable experimental design.

The $\mathcal{I}$-Net parameters were optimized for each model independently, to get the best performance for each approach.

However, the parameters controlling the surrogate function family can obviously not be set equally for Logistic Regression and Decision Trees surrogate functions types. They should not be optimized independently either, as this may lead to bad interpretability of the Decision Tree, as already discussed. Instead, the hyperparameters of the Decision Tree should be set in a way, that the number of decision nodes matches approximately the number of Logistic Regression Parameters (see 4.2).

The number of parameters of a Logistic Regresssion model is fixed at $m = n + 1$, where n is the number of features and m is the number of parameters (bias and weights). Constraining the number of decision nodes in a Decision Trees can be done by controlling its depth.

This leads to the following formula for setting the depth of the Decision Trees generated by the $\mathcal{I}$-Net instance:

$$\begin{aligned} n + 1 &\approx 2^d - 1 \\ \Leftrightarrow d &\approx \log_2(n+2) \end{aligned}$$

$$\text{where} \quad \begin{aligned} d &= \text{depth} \\ n &= \text{number of features} \end{aligned}$$

The max depth hyperparameter is equal to the depth of the tree in the context of this $\mathcal{I}$-Net instance, i.e. the Decision Tree will always be fully grown. For more details on the relationship between the max depth parameter and the number of decision nodes please refer to section 2.2.

Given this formula, d is set to {n=5: 3, n=10: 4, n=20: 4}. Please see appendix B for a detailed overview of all hyperparameter used for this experiment.

Plain Decision Trees

The same settings for the maximum depth were applied to the plain Decision Tree model as well. Plain Decision Trees were trained using the global surrogate modelling approach [27]. For each $(X, Y') \in \mathcal{D}'_{test}$, an independent Decision Tree classifier was trained by using 4'250 samples for training and 750 samples for testing. Then, the mean of the scores for each model was reported. The implementation is supported by the scikit-learn DecisionTreeClassifier API [4].

Experiments with added noise

To provide a better understanding of how well the trained models can generalize the mapping function $\mathcal{I}$, the experiments were repeated preserving the same parameter settings, but with added noise. The noise was added to some training data $(X, Y') \in \mathcal{D}'_{train}$ for the interpretation net through the $noise$ parameter of the $\mathcal{I}$-Net custom loss function (algorithm 3). The noise was implemented the same way for the training processes of the other models. Using this method, the probabilities are flipped for a fraction of the training samples, parameterized by $noise$.

With this type of noise, we get a measure for how well the two different Interpretation Networks generalize and manage to abstract from the actual training data to the task, which should be learned: To provide a mapping function from arbitrary neural networks to their surrogate function of a specific type. If the performance of the neural network with noise is significantly lower than without noise, the neural network has not only learned the actual function, but also the noise, i.e. it has generalized poorly. However, if the model with noise provides similar performance, it is able to generalize and approximates the actual function well.

The experiments with noise were run with $noise = 0.1$, i.e. ten percent of the training data labels for the models have been flipped.

4.3 Experimental Results

Overall, this evaluation covers six different experiments on four different models. A full list of results is provided in appendix C. The fidelity scores and ROC-AUC scores can be viewed also in table 4.1 or 4.2 respectively.

In general, the results show that the performance of $\mathcal{I}$-Nets for Logistic Regression and plain Logistic Regression is better than that of the Decision Tree models. Considering the fidelity score, the plain Logistic Regression model achieves the highest score of all compared models in 5/6 experiments and the $\mathcal{I}$-Net for logistic regression in 1/6 experiments. For the ROC-AUC, this was the case for both in 3/6 experiments. Taking all scores into account, the plain Logistic Regression performs best in most cases, while the $\mathcal{I}$-Net Logistic Regression model offers higher performance for $n = 5$ with added noise. The ROC-AUC score in this case is about 0.1 points higher than with plain Logistic Regression. In general, the performance of the $\mathcal{I}$-Net Logistic Regression model does not decreases through added noise. However, performance suffers for large n, i.e. $n = 20$, drastically, especially in experiments without noise. In this case, the ROC-AUC score indicates the model near the performance of a random guess, which can be intuitively seen in the ROC curves (figure 4.4) as well. This goes along with a very low fidelity score of 0.26, which is the consequence of a low recall (0.19), while precision being at 0.5. The recall for the same experiment with added noise is considerable higher (0.54), leading to a higher fidelity score as well. In this case, the ROC-AUC score does not change significantly.

Another thing to note is that the $\mathcal{I}$-Net for Logistic Regression performance for $n = 20$ with noise is significantly higher than for n=20 without noise. The fidelity score is 0.52 and 0.26, respectively.

The global surrogate models, plain Logistic Regression and plain Decision Trees, provide fairly consistent performance for different n, with 0.06 being the decrease in fidelity from $n = 5$ to $n = 20$ for Decision Trees and 0.006 for Logistic Regression. However, the addition of noise has an impact on the plain Logistic Regression and plain Decision Tree models. Experiments report a decrease of around 0.07 to 0.1 for both F1 and ROC-AUC metrics. This performance decrease can be seen for $\mathcal{I}$-Nets for Decision Trees as well, but it is considerably smaller.

Nevertheless, the $\mathcal{I}$-Net for Decision Trees generally does not perform well in these experiments. The fidelity metric indicates reasonable results for n=5 and n=10 with values between 0.6 and 0.7. However, the delta of precision and recall is constantly high at around 0.5 and the ROC-AUC score is worse than or similar to that from a random guess for n=10 and n=20 (while indicating competitive performance for n=5). This leads to distinct ROC-curves for different n (see figure 4.4). While the curve for n=5 shows an above-random classifier, the curve for n=10 indicates a below-random classifier and for n=20, the model behaves very similar to a random classifier. In general, the $\mathcal{I}$-Net for decision trees performs the worst of all models and performs inconsistent as well. The MCC metric (see appendix C) of the $\mathcal{I}$-Net for Decision Trees is for all experiments at around zero, which indicates

	without noise			*noise* = 0.1		
Experiment	**n=5**	**n=10**	**n=20**	**n=5**	**n=10**	**n=20**
$\mathcal{I}$-Net for Logistic Regression	0.912	0.816	0.259	**0.913**	0.818	0.523
Plain Logistic Regression	**0.955**	**0.946**	**0.948**	0.864	**0.857**	**0.859**
$\mathcal{I}$-Net for Decision Trees	0.619	0.700	0.089	0.597	0.638	0.090
Plain Decision Trees	0.937	0.917	0.870	0.850	0.833	0.796

Table 4.1: Comparison of fidelity scores (best values for each experiment in bold)

that, from this specific score speaking, the predictions of the classifier are not at all correlated with the true test values.

The opposite finding can be stated about the surrogate model techniques, plain Logistic Regression and plain Decision Trees. Both, especially the instance for Logistic Regression, have consistent performance for different n, in comparison to the $\mathcal{I}$-Nets. However, they do perform slightly worse with added noise.

	without noise			*noise* = 0.1		
Experiment	**n=5**	**n=10**	**n=20**	**n=5**	**n=10**	**n=20**
$\mathcal{I}$-Net for Logistic Regression	**0.973**	0.906	0.501	**0.971**	**0.904**	0.539
Plain Logistic Regression	0.954	**0.946**	**0.948**	0.864	0.857	**0.859**
$\mathcal{I}$-Net for Decision Trees	0.721	0.288	0.484	0.686	0.274	0.481
Plain Decision Trees	0.937	0.917	0.871	0.850	0.834	0.796

Table 4.2: Comparison of ROC-AUC scores (best values for each experiment in bold)

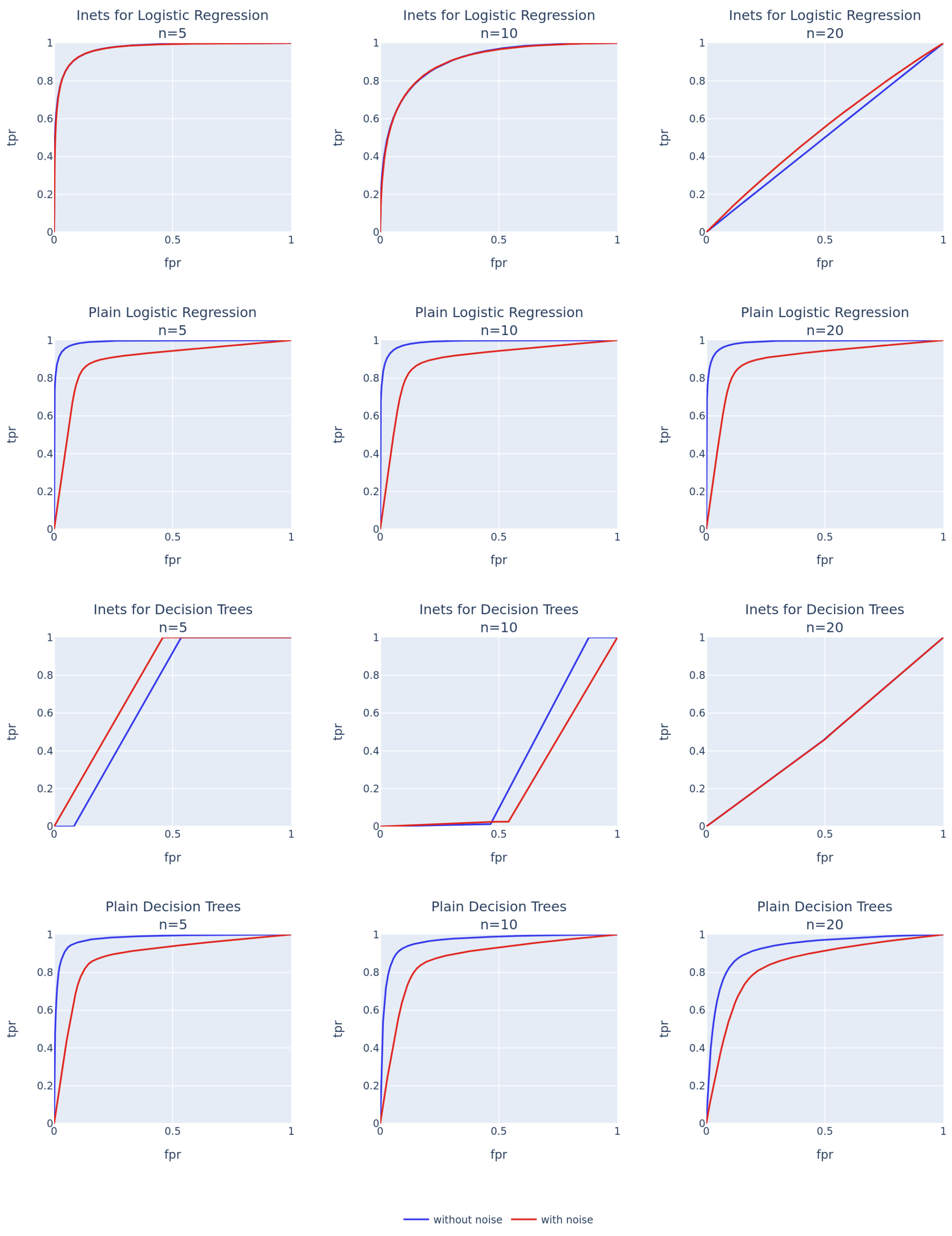


Figure 4.4: ROC curves

Chapter 5

Conclusion

Given the metrics, the following observations can be made:

- Plain Logistic Regression performed the best overall and is very consistent
- The $\mathcal{I}$-Net instance for Logistic Regression has similar performance compared to plain Logistic Regression, except for $n = 20$
- The performance of the $\mathcal{I}$-Net for Logistic Regression does not degrade with random training noise
- The $\mathcal{I}$-Net for Decision Trees does perform the worst of all models, especially for $n = 10$ and $n = 20$

5.1 Discussion

Plain Logistic Regression performed very well throughout the experiment. This was expected, as this model type can access the most amount of information during training time, as it is trained directly on the λ-Net training data. Therefore, performance can be considered best throughout 5/6 experiments (given the fidelity scores). Furthermore, the scores are very consistent and robust for different n and added noise. This is true for the plain Decision Tree models as well. They showed similar performance, with slightly lower scores specifically for larger n, i.e. $n = 20$. In general, training a Decision Tree obviously is more complex compared to a Logistic Regression classifier, which is not necessary given the relatively simple type of functions generated through the Synthetic Data Generation. In general, the make classification function used in the Synthetic Data Generation makes problemsets naturally solvable by Logistic Regression classifiers. They are not optimized for Decision Tree classifiers [29] [4].

The low performance of the $\mathcal{I}$-Net instance for Decision Trees in this experimental setting was not expected. This can probably be solved by hyperparameter optimization and changing implementation details, as this was not the focus of this thesis. Currently the evaluation λ-Nets for this model introduce a bias into the evaluation process, as their predictions are not balanced. The hyperparameter settings are from high importance, as the learning task, finding the function $\mathcal{I}$, is more complex for Decision Tree than for Logistic Regression surrogate functions. The parameter of a Logistic Regression model are forming a smooth decision boundary: Changing a random parameter by a small amount may be relevant for some edge cases, but in general, the modelled function is relatively stable. Only if multiple parameters are changed or one parameter is changed by a large amount, the represented classification function will likely change significantly. This is not true for Decision Trees: Due to the "hard splits", small changes in the split value can lead to completely different decision functions. Small errors in the mapping function will have therefore more impact on the results.

The $\mathcal{I}$-Net for Logistic Regression performed well for smaller n and worse for larger n. The performance decrease for larger n may be because of the higher complexity. Therefore, more hyperparameter optimization and larger datasets can lead to improvements (see section 5.3).

Another anomaly to note is the higher score for $n = 20$ with added noise in comparison to $n = 20$ without noise. This is unexpected. Given $n = 20$, the fidelity score (at 0.26) indicates performance worse than a random guess. One explanation could be overfitting. In comparison, with added noise, the classifier will overfit more on the random noise and less on the function of the training samples itself. This can explained the higher score for the same experiment with added noise, as this score (at 0.52) has similar performance as a random guess.

Besides from that, the $\mathcal{I}$-Net instance can be trained on the desired function and delivers sufficient performance for the interpretation of arbitrary λ-Net. The experiments with added noise show that, comparing the $\mathcal{I}$-Net instance to plain Logistic Regression, it is robust to the addition of training noise. As discussed in 4.2, this shows, that the $\mathcal{I}$-Net generalizes better for arbitrary λ-Nets than plain Logistic Regression. This is of practical advantage as λ-Net functions presumably diverge widely for real world use cases.

5.2 Summary

This thesis proposed a new $\mathcal{I}$-Net instance using a Logistic Regression surrogate model. It was empirically evaluated against the existing $\mathcal{I}$-Net for Decision Trees, proposed by Marton et al. [24], and against global surrogate modelling for both, Decision Trees and Logistic Regression.

The evaluation results lead to three main learnings from the evaluation with regards to the goals of this thesis.

First, the $\mathcal{I}$-Net instance for Logistic Regression proposed in 3.2 can be used for explaining neural networks, if the number of features remains small, e.g. n=5 or n=10. In these cases, the model achieves on par performance both with $\mathcal{I}$-Nets for Decision Trees and plain Logistic Regression. It is also robust against label noise on the training data.

Second, the $\mathcal{I}$-Net for Logistic Regression performance suffers if applied to problems with larger feature set, e.g. n=20. It requires further investigation, whether this is immanent to the design of the instance or can be solved through more hyperparameter optimization (see section 5.3)

Third, the $\mathcal{I}$-Net for Decision Trees has suffered from learning the desired function in this specific evaluation setting. This could be because of the harsh decision boundary build up by Decision Tree classifiers in general. Performance is good with direct supervision (i.e. in the case of plain Decision Trees), but small errors in the $\mathcal{I}$-Net mapping function lead to large errors of the surrogate function. More extensive hyperparameter tuning presumably will lead to higher scores.

5.3 Future Work

The proposed $\mathcal{I}$-Net instance can be enhanced and evaluated further in the following ways:

First, in addition to the evaluation on synthetic data, the proposed $\mathcal{I}$-Net instance for Logistic Regression should be evaluated against real world problems as well, to measure performance in a more realistic setting. As described in section 1, real world use cases are common and continue to evolve.

Second, it should be investigated, whether extensive hyperparameter tuning can lead to better performance, especially for larger values of n. The experiments in chapter 4 used best practises for the model tuning. However, more iterations of experiments and/or automated hyperparameter tuning, e.g. grid search [4] or greedy search [19] could improve performance further. The latter technique was used by Marton et al. with $\mathcal{I}$-Nets for Decision Trees and thereby proved applicability for $\mathcal{I}$-Net instances in general.

Third, hyperparameter tuning for $\mathcal{I}$-Nets for Decision Trees should be done more extensively, as this was not the focus of this work. However, it became obvious, that this $\mathcal{I}$-Net do require more tuning in order to deliver the expected results.

Fourth, a more generalized λ-Net representation could enable high performance $\mathcal{I}$-Nets for different λ-Net hyperparameters and layouts, especially different numbers of layers, different types of layers and different dimensions. This can be achieved by including more information about the λ-Netarchitecture in its representation. This way, $\mathcal{I}$-Nets could be applied to arbitrary λ-Nets, leading to higher applicability in practise. A first step may be to allow different numbers of nodes for each layer. This could be achieved through dimension reduction [6].

Fifth, as of now, several $\mathcal{I}$-Net instances with different surrogate function types have been proposed. However, all of them assume a neural network as λ-Net function. Other machine learning algorithms are considered being a black box as well, e.g. support vector machines (SVMs) [15]. SVMs are from high interest specifically in combination with a Logistic Regression surrogate function type, as SVMs and Logistic Regression are to some extend related with SVMs can be seen as a more complex generalization of Logistic Regression [34].

Bibliography

[1] Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15(3):223–250, 1994.

[2] Nick Bostrom and Eliezer Yudkowsky. *The ethics of artificial intelligence*, pages 316–334. Cambridge University Press, 2014.

[3] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[5] Alan Bundy and Lincoln Wallen. Breadth-first search. In *Catalogue of artificial intelligence tools*, pages 13–13. Springer, 1984.

[6] Miguel A Carreira-Perpinán. A review of dimension reduction techniques. *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09*, 9:1–69, 1997.

[7] Shaoxia Chen, Greg McMullan, Abdul R Faruqi, Garib N Murshudov, Judith M Short, Sjors HW Scheres, and Richard Henderson. High-resolution noise substitution to measure overfitting and validate resolution in 3d structure determination by single particle electron cryomicroscopy. *Ultramicroscopy*, 135:24–35, 2013.

[8] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13, 2020.

[9] François Chollet et al. Keras. `https://keras.io`, 2015.

[10] Miruna-Adriana Clinciu and Helen Hastie. A survey of explainable ai terminology. In *Proceedings of the 1st Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence (NL4XAI 2019)*, pages 8–13, 2019.

[11] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pages 127–148. Springer-Verlag, 2003.

[12] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[13] Thomas D Grant and Damon J Wischik. Show us the data: Privacy, explainability, and why the law can't have both. *Geo. Wash. L. Rev.*, 88:1350, 2020.

[14] Isabelle Guyon. Design of experiments of the nips 2003 variable selection benchmark. In *NIPS 2003 workshop on feature extraction and feature selection*, volume 253, page 40, 2003.

[15] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[16] Jerónimo Hernández-González, Inaki Inza, and Jose A Lozano. Weak supervision and other non-standard classification problems: a taxonomy. *Pattern Recognition Letters*, 69:49–55, 2016.

[17] Andreas G. F. Hoepner, David McMillan, Andrew Vivian, and Chardin Wese Simen. Significance, relevance and explainability in the machine learning age: an econometrics and financial data science perspective. *The European Journal of Finance*, 27(1-2):1–7, 2021.

[18] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[19] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD interna-*

tional conference on knowledge discovery & data mining, pages 1946–1956, 2019.

[20] Vlado Keselj. Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6, 2009.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Yi Lin. A note on margin-based loss functions in classification. *Statistics & probability letters*, 68(1):73–82, 2004.

[23] Sascha Marton, Stefan Lüdtke, and Christian Bartelt. Explanations for neural networks by neural networks. *Applied Sciences*, 12(3):980, 2022.

[24] Sascha Marton, Stefan Lüdtke, Christian Bartelt, Andrej Tschalzev, and Heiner Stuckenschmidt. Explaining neural networks without access to training data. *arXiv preprint arXiv:2206.04891*, 2022.

[25] Christoph Meinel and Martin Mundhenk. Mathematische Grundlagen der Informatik, 2002.

[26] Telmo Menezes and Camille Roth. Symbolic regression of generative network models. *Scientific Reports*, 4(1), sep 2014.

[27] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[28] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

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

[30] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[31] Lior Rokach and Oded Maimon. Decision trees. In *Data mining and knowledge discovery handbook*, pages 165–192. Springer, 2005.

[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[33] Kent A Spackman. Signal detection theory: Valuable tools for evaluating inductive learning. In *Proceedings of the sixth international workshop on Machine learning*, pages 160–163. Elsevier, 1989.

[34] Kevin Swersky. Support vector machines vs logistic regression. *Toronto: University of Toronto, CSC2515 Tutorial*, 2013.

[35] Tea Tušar. Optimizing accuracy and size of decision trees. In *Sixteenth International Electrotechnical and Computer Science Conference-ERK*, pages 81–84, 2007.

[36] Xizhao Wang, Yanxia Zhao, and Farhad Pourpanah. Recent advances in deep learning. *International Journal of Machine Learning and Cybernetics*, 11(4):747–750, 2020.

[37] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, Philip S Yu, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[38] Jianlong Zhou, Amir H Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics*, 10(5):593, 2021.

Appendix A

Program Code / Resources

The source code used for chapter 4 can be reached at
https://github.com/p-koenig/2022-04_BA.git

Appendix B

Hyperparameter settings

Hyperparameter	**Value**
Synthetic Data Generation	
number of datasets	*45'000*
samples per dataset	*5'000*
number of features (n)	*{5, 10, 20}*
randomness control (s)	n*=5: 2,* n*=10: 5,* n*=20: 10*
Lambda Networks	
validation split	*0.15*
batch size	*64*
batch normalization	*yes*
layers	n*=5:[50],* n*=10: [50],* n*=20: [100]*
dropout	*none*
loss	*mae*
activation hidden layers	*swish*
optimizer	*adam*
learning rate	*0.001*
epochs	*500*
early stopping	*yes*
Interpretation Network	
validation split	*0.15*
test split	*0.1*
batch size	*256*
batch normalization	*no*
layers	*[2048, 1024, 512]*
dropout	*[0.5, 0.5, 0.5]*
loss	*custom loss*
activation hidden layer	*swish*
optimizer	*adam*
learning rate	*0.001*
epochs	*1000*
early stopping	*yes*

Table B.1: Hyperparameter settings $\mathcal{I}$-Nets for Logistic Regression

Hyperparameter	**Value**
Synthetic Data Generation	
number of datasets	*10’000*
samples per dataset	*5’000*
number of features (n)	*{5, 10, 20}*
Lambda Networks	
validation split	*0.15*
batch size	*64*
batch normalization	*yes*
layers	n*=5:[50],* n*=10: [50],* n*=20: [100]*
dropout	*none*
loss	*mae*
activation hidden layers	*ReLU*
optimizer	*adam*
learning rate	*0.001*
epochs	*500*
early stopping	*yes*
Interpretation Network	
validation split	*0.25*
batch size	*256*
batch normalization	*yes*
layers	*[1792, 512, 512]*
dropout	*[0.5, 0.5, 0.5]*
loss	*custom loss Marton et al.*
activation hidden layer	*sigmoid*
optimizer	*adam*
learning rate	*0.001*
epochs	*1000*
early stopping	*yes*
Surrogate function family	
maximum depth	*n=5: 3, n=10: 4, n=20: 4*

Table B.2: Hyperparameter settings $\mathcal{I}$-Nets for Decision Trees

Appendix C

Further Experimental Results

Metric	Description
Precision (PRE)	see section 4.1
Recall (REC)	see section 4.1
Fidelity (FID)	see section 4.1 Please note, that the fidelity score is calculated as the mean F1 score from all evaluation datasets. Precision and Recall are calculated in the same way. A F1 score calculated post hoc from PRE and REC using the formula in 4.1 will lead to a different value.
Matthews correlation coefficient (MCC)	MCC is also known as Phi coefficient. Can be used as an alternative to the F1 score for binary classification. Score is in range [-1, 1], with 1 predictions and truth is perfectly correlated. A score of 0 indicates that the classifier is a random guess and -1 indicates perfect negative correlation. [8]
Accuracy (ACC)	The ratio of correctly assigned samples
ROC-AUC (ROC)	see section 4.1

Table C.1: Overview of the metrics used in table C.2

Experiments without noise	**PRE**	**REC**	**FID**	**ACC**	**MCC**	**ROC**
n=5						
$\mathcal{I}$-Net for Logistic Regression	0.914	0.913	0.912	0.912	0.827	**0.973**
Plain Logistic Regression	0.954	**0.956**	**0.955**	**0.955**	**0.910**	0.954
$\mathcal{I}$-Net for Decision Trees	**1.000**	0.464	0.619	0.464	0.000	0.721
Plain Decision Trees	0.940	0.936	0.937	0.937	0.876	0.937
n=10						
$\mathcal{I}$-Net for Logistic Regression	0.837	0.820	0.816	0.821	0.656	0.906
Plain Logistic Regression	0.947	**0.946**	**0.946**	**0.946**	**0.893**	**0.946**
$\mathcal{I}$-Net for Decision Trees	**0.997**	0.549	0.700	0.549	0.000	0.288
Plain Decision Trees	0.920	0.915	0.917	0.917	0.835	0.917
n=20						
$\mathcal{I}$-Net for Logistic Regression	0.503	0.188	0.259	0.501	0.002	0.501
Plain Logistic Regression	**0.948**	**0.949**	**0.948**	**0.948**	**0.897**	**0.948**
$\mathcal{I}$-Net for Decision Trees	0.048	0.049	0.089	0.479	-0.010	0.484
Plain Decision Trees	0.871	0.871	0.870	0.871	0.743	0.871

Experiments with *noise* = 0.1	**PRE**	**REC**	**FID**	**ACC**	**MCC**	**ROC**
n=5						
$\mathcal{I}$-Net for Logistic Regression	0.916	**0.913**	**0.913**	**0.912**	**0.829**	**0.971**
Plain Logistic Regression	0.865	0.864	0.864	0.864	0.729	0.864
$\mathcal{I}$-Net for Decision Trees	**1.000**	0.441	0.597	0.441	0.000	0.686
Plain Decision Trees	0.850	0.851	0.850	0.850	0.701	0.850
n=10						
$\mathcal{I}$-Net for Logistic Regression	0.837	0.825	0.818	0.823	0.660	**0.904**
Plain Logistic Regression	0.856	**0.859**	**0.857**	**0.857**	**0.714**	0.857
$\mathcal{I}$-Net for Decision Trees	**1.000**	0.479	0.638	0.479	0.000	0.274
Plain Decision Trees	0.836	0.832	0.833	0.834	0.669	0.834
n=20						
$\mathcal{I}$-Net for Logistic Regression	0.529	0.535	0.523	0.529	0.059	0.539
Plain Logistic Regression	**0.858**	**0.860**	**0.859**	**0.859**	**0.717**	**0.859**
$\mathcal{I}$-Net for Decision Trees	0.048	0.500	0.090	0.464	-0.010	0.481
Plain Decision Trees	0.796	0.798	0.796	0.796	0.594	0.796

Table C.2: Further experimental results (best score for each experiment in bold)

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 12.07.2022 Unterschrift