Smart Computing Project G3: Report

Endlicher, Erik
erik.endlicher@uni-rostock.de
217203766
ee073 [*]

Kreikemeyer, Justin Noah
justin.kreikemeyer@uni-rostock.de
217203532
jk942 [*]

Krüger, Tom
tom.krueger@uni-rostock.de
221202286
vy4447 [*]

Haghzad, Mahla
mahla.haghzad@uni-rostock.de
220202837
mh1187 [*]

Zech, Lukas
lukas.zech@uni-rostock.de
217204278
lz071 [*]

March 18, 2022

Abstract

Artificial Neural Networks are a powerful tool for solving complex machine learning problems. However, this power comes at the cost of explainability, as the decisions of large networks are not comprehensible by humans. This is problematic for some application domains, e.g. medicine. To tackle this problem, rule extraction was proposed.

This projects aim is to implement an approach for rule extraction, so that it can be compared to previous implementations. To ensure comparability, the algorithm should be tested on existing data sets and neural networks.

[*]Master Informatik

Contents

1 Team Organisation (D1)

In this chapter, we give some information on the ways in which we collaborated.

1.1 Communication and Team Work

We have chosen discord or real meetings in the student council room or the Mensa as our communication tools. Later, due to pandemic-related restrictions, we mainly relied on Discords video chat.

Additionally, for collaborating on the implementation, we used GitLabs issue-tracker and labels, which organizes the issues in a kanban-like way. We created a merge request for issues and peer-reviewed the code before merging.

1.2 Meetings

For every meeting a protocol was created, which adhered to a template. In the main section, the most important discussions and conclusions were outlined. At the end, the next steps and distribution of work were detailed. At the end of each meeting, the identified tasks were distributed.

2 Goals (D2)

In this section, we define the project goals, which are grounded on a formal review of existing literature and the resulting snapshot of the state of the art.

2.1 Literature Review (D2.a)

The first and arguably most important step of any (scientific) project is to assess the current state of the art. This facilitates the reasonable definition of requirements, so that they are actually achievable and sets a baseline for performance. In this section, we describe the process of collecting and filtering the literature and extracting relevant information from it (Section 2.1.1). In Section 2.1.2, we present the state of the art of the projects domain of research.

2.1.1 Review Process

Our review process follows [TP15] and is structured into six *steps*:

1. formulate the problem and identify the associated main technical terms
2. formulate search queries based on those terms
3. collect all the results
4. pre-filter the literature based on title and abstract
5. assess the quality of the remaining literature and extract the relevant data
6. analyze and compare the features (collected data) to identify final contenders

As a *first step*, we identified recent and historic reviews on the topic "rule extraction from neural networks" [ADT95; AK12; Hai16]. From those, we were able to identify a widely used taxonomy with two axes: type of extraction (decompositional, pedagogical and electic) and rule type (if-then, fuzzy, . . .)[1].

This taxonomy formed the basis for our search terms. As an additional constraint, we only focused on pedagogical approaches, as there is preexisting work on a decompositional approach. Given the narrow scope, coming up with search terms presented some difficulties: We had to identify large enough niches inside the domain of pedagogical rules extraction for neural networks. In other words, we had to find sub-fields of pedagogical rule extraction, which is a sub-field of rule extraction for neural networks, which in turn is a sub-field of machine learning. The search terms identified are:

- **pedagogical rule extraction**
  We are looking for any papers that adapt this term to describe an algorithm which is independent on the network structure.

[1]Note that the taxonomy given in [ADT95] has five axes. However, at this early stage of the review it seemed that the two axes presented here were the most universal ones

- **black-box rule extraction**
  The term black-box is commonly associated with pedagogical rule extraction, as the neural network is viewed as a black-box.

- **reverse engineering**
  This term is also associated and closely related to black-box.

- **sampling approach**
  This was one general technique discussed in [Hai16; SAG99], which is closely related to pedagogical approaches. The idea is to sample outputs of the network given some input samples and make conclusions about the networks internal behavior.

- **fuzzy rule extraction**
  This is a query regarding the second taxonomy axes. As most approaches seemed to be based on if-then rules, we wanted to include fuzzy rules.

- **validity interval analysis algorithm**
  According to [ADT95], this seems to be one of the first approaches to pedagogical rule extraction.

In order to retrieve more relevant results, these queries were combined with terms like “rule extraction” and “neural network”. The seven search engines used and the exact queries made for each can be found in Section A.1. This concludes the *second step* in the overview.

In the *third step*, the results were collected in a large table, and for each entry, title, author, year of publication, url/doi, search query, search engine, and the date of search were stored. This allowed the semi-automatic filtering of results, which reduced the time spent on manual filtering in the *fourth step*.

The automatic part was done using `python` and the popular `pandas` module for dataset management. First, a column was added indicating whether the entry is a duplicate, based on the combination of title and year (`duplicate`). Then, all entries were marked as `keywords`, which contained at least one relevant keyword, such as “rule extraction” or “pedagogical”, in its title. For this, a regular expression was used. All papers containing a word like “survey” or “review” were also marked and saved in an extra table for later reference. Last, but not least, all entries with a publication year smaller than 2005 were marked for exclusion (`year_limit`). This decision was based on an analysis on the whole dataset, shown in Figure 1. From the boxplot it is apparent, that the significant mass of papers was published later than 2005. Further analysis with a histogram implied, that the year 2005 marks the beginning of one of two major publication waves. The column `include` indicates, based on the previous columns, whether the paper should be included (as determined by the automatic filtering). Finally, some columns were added for manual remarks on whether the paper is `accessible`, and the `abstract` is promising.

The automatic filtering yielded an extended table, which was then manually checked and completed. To prepare for the fifth step, we extracted the URLs/DOIs of the entries marked for exclusion, and imported them in a Zotero[2] shared library.

[2] https://www.zotero.org/

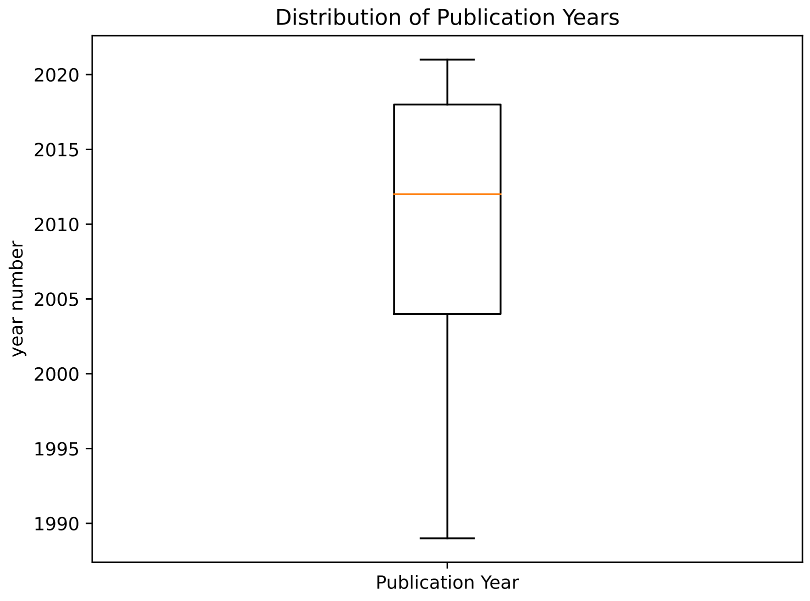


Figure 1: Distribution of publication years among all papers resulting from the search queries. The plot is produced using the `Matplotlib` python module on the table obtained from the third step.

This way, we were able to automatically download the PDFs of most papers, and generate the `biblatex` file for this document.

For the *fifth step*, we defined a couple of criteria/features, which described the remaining papers in terms of properties of the approach, evaluation results and quality. The concrete criteria can be found in Section A.2. This time, a google sheets table was used for easy collaboration. During the evaluation, we identified two additional papers that met our inclusion criteria but were not included in the table of the fourth step. Our evaluation is given in Section A.3.

We assessed all of the 46 papers against our evaluation criteria and marked the five most promising for the final selection of *step six*.

X-TREPAN [KZ15] is an extension of the well known TREPAN [CS95] algorithm from 1995. It is a purely pedagogical approach to extract a decision tree from a feed forward neural network. From the paper [KZ15] it is not clear how X-TREPAN extends the TREPAN algorithm. However, the authors test and describe the original TREPAN algorithm on different datasets. TREPAN views the extraction as an inductive learning problem and uses a network as an oracle that is able to answer queries during the learning process. As input, the algorithm receives the trained network along with training data labeled by the network. There exist alternatives for defining the splitting criteria in the nodes of the tree. The rules for classification can be simply extracted from the final tree. Detailed work and implementations exist for the TREPAN methodology [CS95].

HYPINV [SW07] is a pedagogical method to explain a neural network by using inversion. It attempts to calculate the ANN input which produces a desired output. The output of the ANN should be binary but the inputs can be binary or continuous. The network decision boundary is expressed in the form of a rule base, made of conjunctions and disjunctions of different rule antecedents. Most parts of the given algorithm in the paper seem to be implementable. However, the algorithm needs a network inversion technique to find the projection of a given point on the network decision boundary. The paper refers to different techniques for achieving this, for instance an evolutionary algorithm. Suggestions for rule simplification and to include multiple classes are given. The approach is tested and analyzed on different datasets.

ALPA [JM15] (Active Learning-Based Pedagogical rule extraction) is an active-learning approach, which questions a black-box model and then uses rule induction on the generated samples. As such, it can be used on any model and with any rule induction technique and is ambiguous to the type of explanation extracted. However, the paper focuses on the classification task. It is based on TREPAN [CS95], in that it uses the black-box model as an oracle to re-label data points and improve the fidelity of the generated rules. It combines this with the idea of ALBA [MBV09], a decompositional approach specific to SVMs, which not only uses existing data points, but uses the black-box model to generate new labeled data points near the decision boundary to further improve fidelity and accuracy. It has a complexity of $\mathcal{O}(C * N * m^5)$ or $\mathcal{O}(N^2 * m + N * C)$ (depending on the algorithm used for identifying points near the decision boundary, so called valley points), where C is the time the black-box model takes to make a prediction, N is the number of valley points and m is the number of input dimensions. In an extensive evaluation on WEKA, they show that ALPA is able to generate comprehensible rule sets with a high fidelity and accuracy.

G-Rex [JKN10; Joh07] (Genetic Rule EXtraction) is a technique based on genetic programming. The G-Rex algorithm can be used on an arbitrary model and generates an output based on a user-defined grammar. Its performance and complexity heavily depend on the choice of genetic programming related parameters, like fitness function, cross over, etc. The most important component is the fitness function, as it ensures the fidelity of the rule set, i.e. minimizes its error w.r.t. the model. The paper [JKN10] extends G-Rex by a new fitness function (i.e. fidelity measure), called generalized Brier score. This makes it possible to not only mimic the predictions of the black-box (opaque) model, but also its probability estimations for those predictions. An evaluation on WEKA datasets shows that the proposed improvement outperforms the standard G-Rex utilizing fidelity.

Kennen [OSF19] is an approach for metamodels considering reverse-engineering network hyperparameters that predicts the attribute of a black-box model. By learning the correlation between the black-box neural network attributes and certain patterns in the network's output with respect to certain query inputs, diverse type of information can be extracted by observing its output. The correlation is learned by training a classifier over outputs from multiple models to predict the model attributes. The three metamodel methods introduced in the paper are kennen-o, kennen-I, and kennen-io. They differ in the way they choose the query inputs and interpret the corresponding outputs. The methods were evaluated by implementing on black-box MNIST digit classifier, and it was mentioned that they can extract inner details with great precision and generalizability.

2.1.2 State of the Art

From the in-depth study of the sixth stage-papers above, and the evaluation of the most relevant fifth stage-candidates in Section A.3, we conclude that the field of pedagogical rule extraction is still a very active research field. The proposed solutions all have drawbacks in terms of accuracy, fidelity, comprehensibility, scalability, or generality. Progress seems to be slow, with old methods, such as TREPAN [CS95] and VIA [Thr95], still being cited used for comparison. One could argue that it is difficult to develop novel approaches, given the limitation of not analyzing the model structure.

All pedagogical approaches have in common that they rely on the provided black-box/opaque model to generate new data labels, which is the only option without opening the black-box. Their diversity rather originates from the types and number of queries they make, and how they use the resulting labels. This results in varying accuracy and fidelity among algorithms.

2.2 Requirements (D2.b)

In due consideration of the state of the art, described in Section 2.1.2, we define the requirements for this project.

2.2.1 General approach

Artificial Neural Networks (ANNs) already show very promising results at solving complex machine-learning tasks. However, with regards to how they solve these problems, they are largely still black-boxes. This is often no concern, but in safety-critical domains, for instance medicine and finance, this is unacceptable.

Therefore, this project aims to extract rules out of ANNs that are readable for humans, and thus make these ANNs human-understandable. As mentioned before, three general approaches already exist to solve this problem. This project implements the pedagogical approach, where the network is considered as a black-box, so it is possible to compare it to others. Furthermore, there are two main problem-types that ANNs solve: classification problems and regression problems. Our project only focuses on ANNs that solve classification problems. In order to implement this, we aim to build on an existing framework. This framework has been developed by a previous group, which tried to solve rule extraction through a decompositional approach. Their implementation and documentation can be found in their Git-Repository[3].

We choose to implement the aforementioned ALPA approach [JM15] into the framework. Our literature review showed the advantages of ALPA: a broad range of application and competitive evaluation results. The approach builds on and achieves major improvements over existing ideas like TREPAN and ALBA, indicating its state of the art status. Furthermore, the corresponding paper exhibits

[3]https://git.informatik.uni-rostock.de/mmis/Teaching/2021SS/NEIDI-G5-Rule_Extraction

a high quality, containing many visualizations and algorithmic examples, as well as a link to the Java source code, making analysis of the algorithms and adoption into the framework easier.

2.2.2 Structure of Implementation

We aim to structure our implementation similar to the existing project. In the end we want to have three different modules: Training, Extraction and Evaluation. The Training module takes a given data set and trains a feed-forward Neural Network (FFNN) on it. The ALPA approach [JM15] generally works on all kinds of ANNs, but since the authors use FFNNs in their paper, we also want to use FFNNs for improved comparability. The Extraction module then takes the trained FFNN and extracts logical rules from it.

The reason for this is that most literature focuses on logical rule extraction, thus making it easier to compare our approach to the state of the art. The propositional logic formulas will be given in disjunctive normal form. Lastly, the Evaluation module takes both, the FFNN and the extracted rules, to evaluate the extracted rules with regards to the evaluation metrics. The metrics we use are explained in the next section.

2.2.3 Evaluation

Naturally, performance assessment is an important part of any implementation. Similarly to existing solutions, such as the ALPA approach [JM15] and [ADT95], we introduce metrics for evaluation. The first objective metric here is *fidelity*, which describes how close the rule set resembles the black-box network. Furthermore, we introduce an *accuracy* measure on how well the rule set performs on the input data. We therefore aim to achieve high values for these two metrics. Since rule extraction aims to generate a human-comprehensible rule set, the *comprehensibility* of the generated rules should also be discussed. One should keep in mind however, that this is a subjective metric, so targets here can not be clearly defined. Therefore the target can only be described as "comprehensible rules" at this stage. Finally, we aim to implement the framework with high *scalability* and *generality*, so that it is applicable in many scenarios. For achieving this, different experiments need to be defined to assess the performance of the framework for different scales. In order to perform and evaluate experiments, datasets have to be selected. To allow comparison with other approaches, we chose the following known and tested datasets:

- Breast Cancer Wisconsin (`https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28original%29`)
- Iris (`https://archive.ics.uci.edu/ml/datasets/iris`)
- MNIST (`http://yann.lecun.com/exdb/mnist/`)
- UCI Credit Card (`https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`)

2.2.4 Non-functional Requirements

Next to the more formal requirements we also desire our system to be well documented. Providing adequate documentation makes it easier for third parties to use the framework.

We also aim to have the final output, i.e. the evaluation, in a form that makes it comparable to the previous projects of the University of Rostock in this subject area.

Lastly, if there is still time left after achieving the requirements, we also aim to implement the points mentioned in the "Future Work" sections of the previous group's project.

2.2.5 What-If Scenarios

Finally, we want to discuss some "what-if" scenarios, which would lead to changes in the requirements mentioned above in case of unexpected external influences or unexpected discoveries.

Team size reduction would lead to re-balancing the workload. Since this generates more work for individual team members, it is possible that certain requirements can not be met. These could include the use of less datasets for testing or less refinement for the documentation.

Implementation or technical problems require different solutions in case of missing time to solve them. As a fallback for unsolvable problems, we would use WEKA[4], a testing framework that ALPA integrates with. This would still allow us to generate results for comparison with previous scenarios. Similarly, if it turns out that the integration of ALPA into the framework does not work, we would try to make the framework compatible or switch to the just mentioned WEKA.

[4] https://www.cs.waikato.ac.nz/ml/weka/

3 Concept (D3)

Here, we aim to describe how the goals mentioned in 2.2 are going to be conceptualized. The following should be understood as a general plan, which will be detailed/changed while implementing and discovering new information.

As mentioned before, our implementation will be based on the existing pipeline and can be divided into three modules: **Training**, **Rule Extraction** and **Evaluation** (see Figure 2). These modules will be implemented using Python by improving upon the existing modules in terms of configurability, documentation and code-style as detailed later.

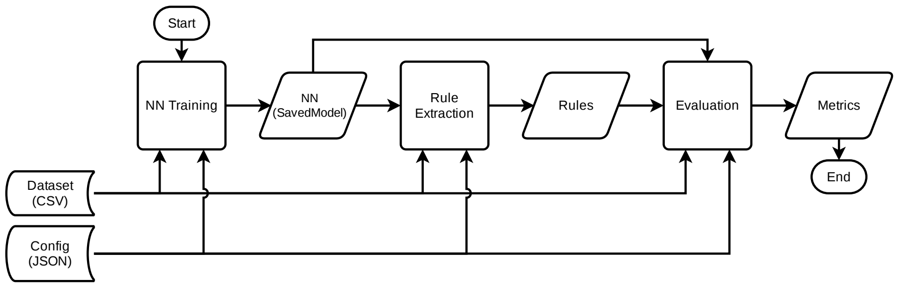


Figure 2: Implementation Pipeline described through the different modules
(This diagram is taken from the previous groups project report)

3.1 Implementation: Training

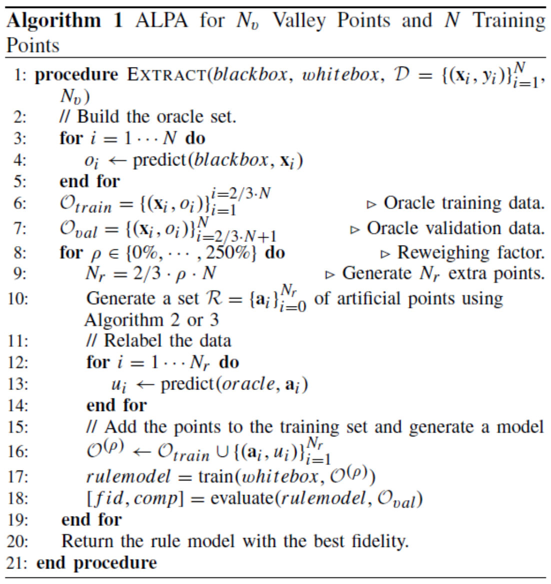
As seen in Figure 2, the Training module takes its configuration as JSON files, while the dataset for the FFNN is given as a CSV. The former here includes information about the structure of the FFNN and general configuration parameters, while the latter gives the dataset to train the FFNN on. The module then takes these parameters and constructs the neural networks and performs initial training of said network. This will be done using the packages Tensorflow and Keras.

3.2 Implementation: Rule Extraction

The only part that needs a completely new implementation is the rule extraction algorithm. Multiple ways might be tried in order: using the existing implementation in Weka, and transcribing the Java implementation to Python. Actually, if both turn out to be possible, they can be made available through the configuration file.

For the first approach, it is necessary to make the compiled Tensorflow FFNN model readable by WEKA, which uses its own FFNN implementation in Java. Preliminary results showed that this might turn out to be intractable to realise. Instead, it might be possible to train the network directly in WEKA based on the supplied configuration. Second, the ruleset has to be exported to a file to be further processed by the pipeline and adhere to the `if precondition then conclusion` style, where the precondition is given as disjunctive normal form.

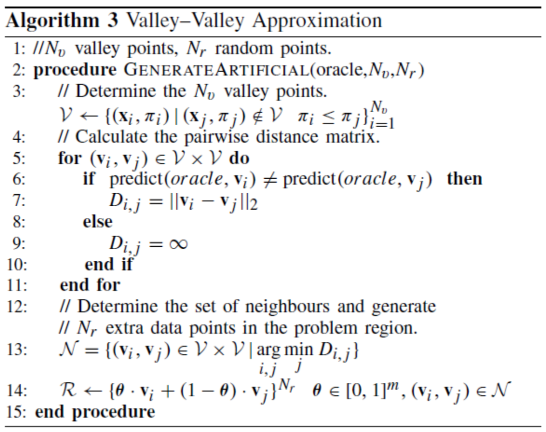
For the implementation of ALPA in Python alongside the existing DeepRED decompositional extraction, we need to select a rule induction technique. We will



Algorithm 1 ALPA for N_v Valley Points and N Training Points

```
1: procedure EXTRACT($blackbox$, $whitebox$, $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, $N_v$)
2:     // Build the oracle set.
3:     for $i = 1 \cdots N$ do
4:         $o_i \leftarrow$ predict($blackbox$, $\mathbf{x}_i$)
5:     end for
6:     $\mathcal{O}_{train} = \{(\mathbf{x}_i, o_i)\}_{i=1}^{i=2/3\cdot N}$              ▷ Oracle training data.
7:     $\mathcal{O}_{val} = \{(\mathbf{x}_i, o_i)\}_{i=2/3\cdot N+1}^{N}$            ▷ Oracle validation data.
8:     for $\rho \in \{0\%, \cdots, 250\%\}$ do                     ▷ Reweighing factor.
9:         $N_r = 2/3 \cdot \rho \cdot N$                       ▷ Generate $N_r$ extra points.
10:        Generate a set $\mathcal{R} = \{\mathbf{a}_i\}_{i=0}^{N_r}$ of artificial points using
           Algorithm 2 or 3
11:        // Relabel the data
12:        for $i = 1 \cdots N_r$ do
13:            $u_i \leftarrow$ predict($oracle$, $\mathbf{a}_i$)
14:        end for
15:        // Add the points to the training set and generate a model
16:        $\mathcal{O}^{(\rho)} \leftarrow \mathcal{O}_{train} \cup \{(\mathbf{a}_i, u_i)\}_{i=1}^{N_r}$
17:        $rulemodel$ = train($whitebox$, $\mathcal{O}^{(\rho)}$)
18:        $[fid, comp]$ = evaluate($rulemodel$, $\mathcal{O}_{val}$)
19:    end for
20:    Return the rule model with the best fidelity.
21: end procedure
```

(a)



Algorithm 3 Valley–Valley Approximation

```
1: //$N_v$ valley points, $N_r$ random points.
2: procedure GENERATEARTIFICIAL(oracle,$N_v$,$N_r$)
3:     // Determine the $N_v$ valley points.
       $\mathcal{V} \leftarrow \{(\mathbf{x}_i, \pi_i) \mid (\mathbf{x}_j, \pi_j) \notin \mathcal{V} \quad \pi_i \leq \pi_j\}_{i=1}^{N_v}$
4:     // Calculate the pairwise distance matrix.
5:     for $(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{V} \times \mathcal{V}$ do
6:         if predict($oracle$, $\mathbf{v}_i$) $\neq$ predict($oracle$, $\mathbf{v}_j$) then
7:             $D_{i,j} = ||\mathbf{v}_i - \mathbf{v}_j||_2$
8:         else
9:             $D_{i,j} = \infty$
10:        end if
11:    end for
12:    // Determine the set of neighbours and generate
       // $N_r$ extra data points in the problem region.
13:    $\mathcal{N} = \{(\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{V} \times \mathcal{V} \mid \arg\min_{i,j} \min_{j} D_{i,j}\}$
14:    $\mathcal{R} \leftarrow \{\theta \cdot \mathbf{v}_i + (1-\theta) \cdot \mathbf{v}_j\}^{N_r} \quad \theta \in [0,1]^m, (\mathbf{v}_i, \mathbf{v}_j) \in \mathcal{N}$
15: end procedure
```

(b)

Figure 3: Pseudocode from the ALPA paper describing the general framework algorithm (a) and the Valley-Valley Approximation algorithm(b)

start by using the existing C50 R library[5] used by DeepRED to extract the decision tree/ruleset. This should allow for an easy comparison.

As seen in algorithm 3(a), ALPA works by generating training data for the rule induction with the help of the black box, using it as an oracle. This allows for more data points to be used for training, which can then be labeled with the black box. The remaining problem then is the generation of new samples, such that training the white box results in better performance. For that, the Valley-Valley Approximation, proposed in [JM15], will be used, further illustrated in algorithm 3(b), since it has the better complexity for our use-case.

3.3 Implementation: Evaluation

For evaluation, we will rely almost completely on the existing code with some minor modifications or additions, like adding a new evaluation measure. Instead of saving it as markdown, we might opt for a PDF or HTML version of the report.

3.4 Improvements of the existing Pipeline

- Augment the configuration possibilities: setting activation functions for hidden layers, more control over rule extraction algorithm and the parameters used, precise control over metrics used in evaluation, (file) type of report generated for evaluation, ...

- In order to support different rule extraction algorithms, implement checks on whether the configured extraction algorithm supports the configured model

[5]https://cran.r-project.org/web/packages/C50/vignettes/C5.0.html

- Remove unused functions or integrate/use them, e.g. `cross_validation.py`, `unit_count.py`
- Refactor weird use of `__init__.py` in the dnnre module
- Refactor the way the pipeline works: make input *and* output files for each module configurable
- If it turns out to be possible, implement a file-based interface to WEKA

3.5 Implementation Result

As the project matured, the initial concept was implemented, but also further extended. The only remaining parts of the original implementation are the implementation of DNNRE and the general modular structure. The configuration capabilities were massively extended and new features, like weight of evidence encoding, convolutional network training and support for multidimensional (image) datasets, were added. Here we put a lot of emphasis on performance. We used *Numpy* whenever possible and the existing R interface, which is based on a *C* implementation of C5.0. We documented all functions and classes with *reST* docstrings.

Usage is possible using a CLI, e.g. running modules and producing output files, which may be processed using another tool, and also through a Python API, e.g. passing instances between modules and extending module classes. More information about the development process and code are available in our Developer Documentation Section A.5. Detailed instructions for users can be found in our User Documentation Section A.4 as well as the API documentation created with *Sphinx*[6]. The new structure of the implementation is detailed in Figure 4.

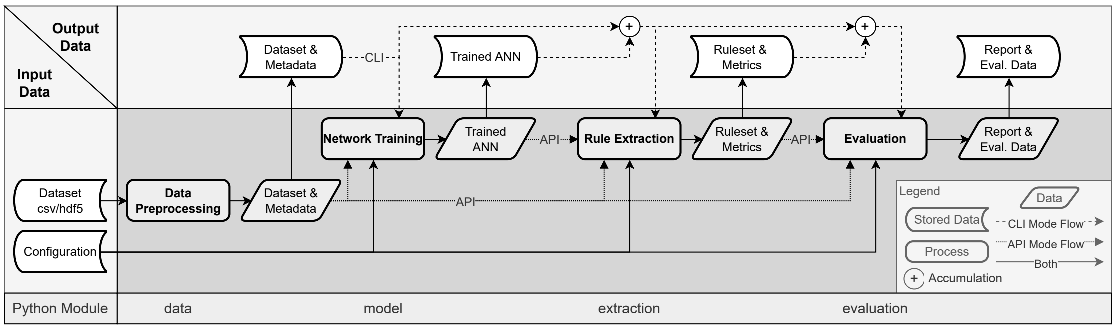


Figure 4: The new and improved Pipeline structure of the implementation.

[6] https://www.sphinx-doc.org/en/master/

4 Evaluation (D4)

In this chapter we perform a hypothesis-driven evaluation of the ALPA approach. We also take comparisons to the existing dnnre implementation into account. It should be noted, that if we talk about comprehensibility in this section, we refer to the degree to which a human can comprehend the generated rules. When we refer to fidelity between two models, we refer to the percentage of matching predictions of all predictions. The accuracy is the percentage of correct predictions of all predictions.

4.1 Hypotheses

To begin our evaluation, we define a number of hypotheses to test. They originate from our initial beliefs, which are based on our experiences and theoretical foundations. For example, we expected results obtained from ALPA to exhibit a lower fidelity than those obtained from DNNRE, given their pedagogical/decompositional nature. The following is the list of hypotheses, numbered (1a) to 5(c), to be tested in the next sections. We use the labels [Falsified] and [Supported] to indicate that the hypothesis could or couldn't be falsified/is supported by the experimental results. The main quantitative results, on which we base these claims, are summarized in Table 1 and Table 2.

1. General
    (a) [Supported] In contrast to DNNRE, the time-performance of ALPA scales better with the amount of hidden network layers.
    (b) [Supported] The generated rules are less accurate than the neural network.
    (c) [Falsified] The fidelity achieved by ALPA-generated rules is smaller than for dnnre-generated rules.
    (d) [Falsified] The ruleset of dnnre will be more comprehensible, containing of less rules.
2. Iris Flower Dataset
    (a) [Supported] Both the network and rule extraction are be able to achieve a high accuracy.
    (b) [Supported] (DNNRE) [Falsified] (ALPA) The rule extraction will converge to a concise ruleset, containing of around 5 rules.
3. MNIST Handwritten Digits Dataset:
    (a) [Supported] In contrast to dnnre, ALPA is able to generate rules for MNIST in reasonable time and with limited resources.
    (b) [Supported] The generated ruleset has a low comprehensibility.
    (c) [Supported] The generated rules reflect the pixels of the hand-written digits.

   (d) [Supported] Of all datasets tested, extracted rules from image data have the worst fidelity.

   (e) [Falsified] The extracted rules from a convolutional neural network have a higher accuracy than the rules from an FFNN.

4. Breast Cancer Wisconsin Dataset

   (a) [Supported] Both the network and rule extraction should be able to achieve a high accuracy, but less than for the Iris dataset.

   (b) [Falsified]The rule extraction will not converge to a concise ruleset.

5. UCI Credit Default Dataset

   (a) [Supported] Encoding the categorical features in the dataset through One-Hot or Weight-of-Evidence (WoE) Encoding raises the accuracy of the network and the rules

   (b) [Falsified] Using WoE instead of One-Hot Encoding increases the accuracy of the neural network

   (c) [Supported] (ALPA) Using WoE instead of One-Hot Encoding increases the fidelity of the rules

Dataset	Network Type	Accuracy Test (Train)
Iris	FF (1)	0.98 (0.99)
MNIST	FF (2)	0.96 (0.98)
	CONV (2)	0.99 (1.00)
Wisconsin	FF (1)	0.98 (0.95)
UCI Credit	FF (1)	0.707 (0.712)
	FF (1) (One-Hot)	0.777 (0.784)
	FF (1) (WoE)	0.774 (0.777)

Table 1: Accuracy of the trained neural networks on the different datasets. The network types are FF, feed-forward (number of hidden layers), and CONV, convolutional (number of convolutional layers)

4.1.1 Hypotheses (1) "General"

1a "ALPA is faster" To test this hypothesis, a number of networks with increasing number of hidden layers were trained on the Iris dataset. Subsequently rules were extracted using both algorithms. The resulting time measurements of single extraction runs can be seen in Figure 5, which provides evidence that ALPA is indeed faster at extracting rules, thus confirming the theoretical complexity results. This experiment was done on a machine with an AMD Ryzen 1600 CPU (up to 3.6 GHz) and 8GB of memory running Linux kernel `5.16.14-arch1-1` in TTY mode (no graphical interface).

Dataset	Algorithm	Acc. Test (Train)	Fid. Test (Train)	#rules
Iris	DNNRE	0.96 (0.96)	0.94 (0.95)	3
	ALPA	0.96 (0.98)	0.94 (0.99)	14
MNIST FF	DNNRE	–.— (–.—)	–.— (–.—)	—
	ALPA	0.90 (0.94)	0.90 (0.95)	601
MNIST CONV	DNNRE	–.— (–.—)	–.— (–.—)	—
	ALPA	0.90 (0.95)	0.90 (0.95)	573
Wisconsin	DNNRE	0.93 (0.93)	0.94 (0.95)	2
	ALPA	0.96 (0.95)	0.96 (1.0)	16
UCI Credit	DNNRE	–.— (–.—)	–.— (–.—)	—
	ALPA	0.707 (0.712)	0.966 (0.982)	151
UCI Credit (One-Hot)	DNNRE	–.— (–.—)	–.— (–.—)	—
	ALPA	0.779 (0.785)	0.982 (0.991)	64
UCI Credit (WoE)	DNNRE	–.— (–.—)	–.— (–.—)	—
	ALPA	0.771 (0.776)	0.989 (0.997)	79

Table 2: Quality of the extracted rules from the neural networks trained on the different datasets. The dash – indicates that the experiment did not finish, because it exceeded our resources in terms of memory and/or time.

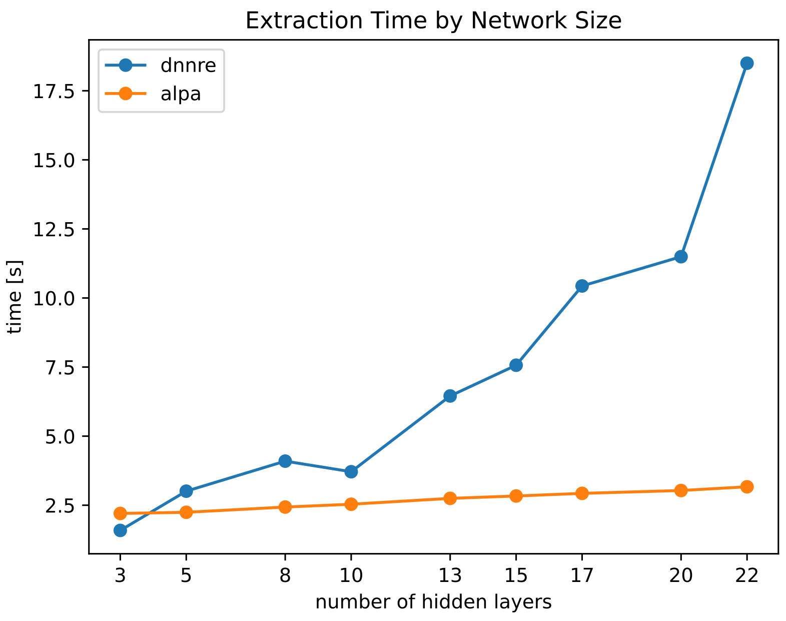


Figure 5: Comparison of extraction time depending on the number of hidden layers. Shown are measurements of a single algorithm execution per datapoint.

1b "Network > Rules" Over all experiments, the network is always more accurate than the extracted rules, which can be seen in Table 1 and Table 2. The difference in accuracy is generally quite small and in the range of 0.01 to

0.1, depending on the algorithm. The largest difference is observed for the image (MNIST) dataset.

1c "ALPA achives lower fidelity" Given the pedagogical nature of ALPA, one could assume that the fidelity of the extracted rules might be smaller than for DNNRE, which is a decompositional approach. However, as evidence presented in Table 2 suggests, this thesis does not hold in the general case.

1d "DNNRE is more comprehensible" As can be seen in Table 2, the DNNRE consistently produces less rules than the ALPA algorithm. However, in addition to the quantitative results shown here, the rules extracted by DNNRE for the Wisconsin Breast Cancer dataset were not complex in terms of the number of rules, but the number of preconditions. This suggests, that the rules constructed by DNNRE might be more comprehensible, but in some situations also just as incomprehensible as ALPA-generated rules, just in a different way.

4.1.2 Hypotheses (2) "Iris Dataset"

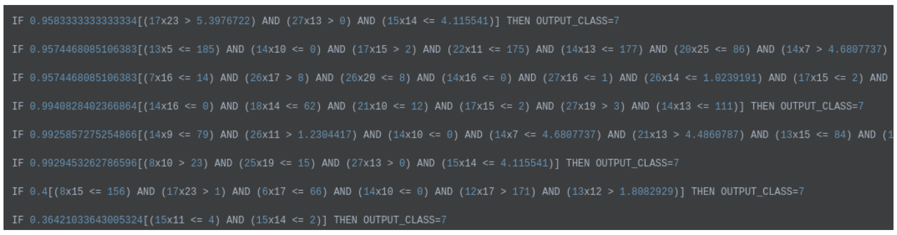
2a In the experiment, the trained network, containing of a single 64-neuron relu layer, achieved an accuracy of 0.98 (0.99) on the test (training) dataset. The rules extracted by ALPA achieved an accuracy of 0.96 (0.98) and a fidelity of 0.94 (0.99). The rules extracted by DNNRE achieved an accuracy of 0.96 (0.96) and a fidelity of 0.94 (0.95). This is evidence that, as described by hypothesis 2a, the network and rule extraction achieve a high accuracy on this dataset. This is no surprise, as it only contains of a few attributes and classes, which are rather easy to separate.

2b Regarding the comprehensibility, it becomes clear that the pedagogical approach of ALPA comes at the cost of more rules. While DNNRE produced only 3 rules, the ALPA algorithm produced 14 rules, which are also more complex, containing of up to 4 terms (compared to 2 for DNNRE). Hence, hypothesis 2b only holds for DNNRE, which produced a very concise ruleset.

4.1.3 Hypotheses (3) "MNIST Dataset"

3a The ALPA algorithm needed about one hour to extract rules from a FFNN (3549 s) or a CONV NN (3544 s). The experiment was conducted on an Intel Core i7 Laptop with 8 GB RAM and 16 GB SWAP memory (Linux). The execution of DNNRE needed more and more RAM and the process was killed after approximately two hours (memkill). The current DNNRE implementation does not support convolutional layers. The successful extraction of rules with ALPA on a common laptop supports hypothesis 3a. The ALPA implementation has a quadratic runtime in the number of training data samples for calculating nearest neighbors, which took about seven minutes. We used an efficient Numpy implementation for the computationally expensive parts. Most of the runtime comes from training, and predicting with, the whitebox in the ALPA loop. We used an R interface to the C implementation of the C5.0 classification model. The DNNRE implementation also uses this interface for parts of the computation. However, the inspection of the NN architecture seems to consume a lot more resources.

3b ALPA generated a set of 573 rules for the CONV NN (601 for FF). The rules are using 342 (343) out of 784 attributes. This means, that about 44% of the pixels are actually necessary to predict the 10 classes. For each class many rules exist, e.g. 54 for the class 7. Each rule has between 2 and 15 terms. Many attributes are needed to predict a single class as well, e.g. 145 (176) pixels for class 7. There are 15 (7) pixels which are used for every class. Figure 6 shows a small subset of the rules that predict class 7. Due to the high number of rules and used attributes



```
IF 0.9583333333333334[(17x23 > 5.3976722) AND (27x13 > 0) AND (15x14 <= 4.115541)] THEN OUTPUT_CLASS=7
IF 0.9574468085106383[(13x5 <= 185) AND (14x10 <= 0) AND (17x15 > 2) AND (22x11 <= 175) AND (14x13 <= 177) AND (20x25 <= 86) AND (14x7 > 4.6807737)
IF 0.9574468085106383[(7x16 <= 14) AND (26x17 > 8) AND (26x20 <= 8) AND (14x16 <= 0) AND (27x16 <= 1) AND (26x14 <= 1.0239191) AND (17x15 <= 2) AND
IF 0.9940828402366864[(14x16 <= 0) AND (18x14 <= 62) AND (21x10 <= 12) AND (17x15 <= 2) AND (27x19 > 3) AND (14x13 <= 111)] THEN OUTPUT_CLASS=7
IF 0.9925857275254866[(14x9 <= 79) AND (26x11 > 1.2304417) AND (14x10 <= 0) AND (14x7 <= 4.6807737) AND (21x13 > 4.4860787) AND (13x15 <= 84) AND (1
IF 0.9929453262786596[(8x10 > 23) AND (25x19 <= 15) AND (27x13 > 0) AND (15x14 <= 4.115541)] THEN OUTPUT_CLASS=7
IF 0.4[(8x15 <= 156) AND (17x23 > 1) AND (6x17 <= 66) AND (14x10 <= 0) AND (12x17 > 171) AND (13x12 > 1.8082929)] THEN OUTPUT_CLASS=7
IF 0.36421033643005324[(15x11 <= 4) AND (15x14 <= 2)] THEN OUTPUT_CLASS=7
```

Figure 6: Subset of rules produced by ALPA on MNIST to predict class 7 (CONV).

it is very difficult to interpret the rules. This shows that the ruleset has a low comprehensibility (hypothesis 3b).

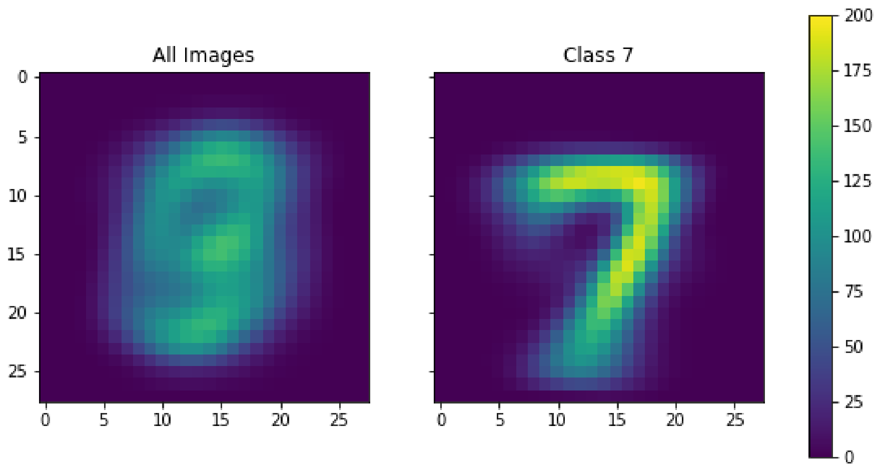


Figure 7: The average pixel intensity of the MNIST dataset.

3c Figure 7 displays the intensity of the pixels in the MNIST dataset. The color shows the average value of each pixel over all samples. To investigate whether the generated rules reflect these pixels we created Figure 8. The heatmap shows the number of times an attribute is in the premise of the rules. A comparison of the pixel intensity over all samples and the attribute counts over all rules (left side of the figures) makes it clear that the rules cover the same areas as the images of the numbers. By comparing the pixel intensity and attribute count for class 7 it gets clear that the pixels of an individual digit are not reflected in the rules for that class. It looks more like the rules use a large number of pixels for each of the classes. For the distinction of the classes, pixels that must have a very low gray value occur frequently in the premises. It is also noticeable that a few pixels are used for all classifications and occur very frequently in the premises. This could be due to the fact that our whitebox first generates a decision tree for the rule extraction and that these pixels are located very high up in this tree.

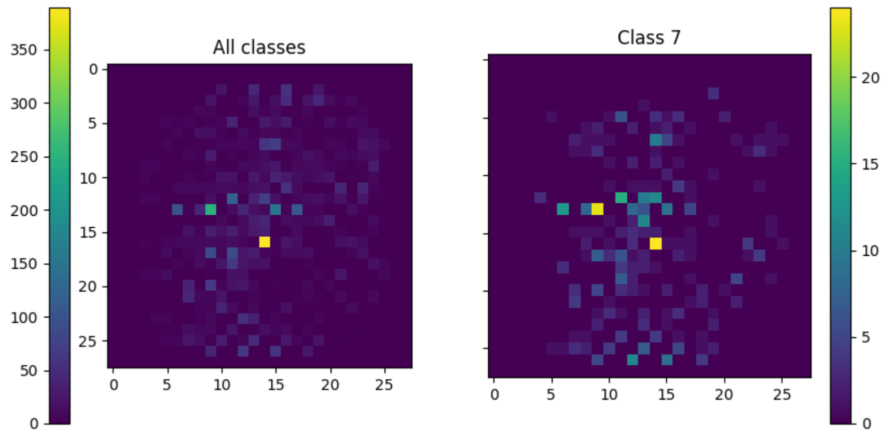


Figure 8: Heatmap for the rules generated by ALPA for MNIST (CONV). It shows the count of attributes for the premises of the rules (pixels). The left side shows the accumulated values over all classes and the right side for class 7 only.

3d For both the FF and CONV NN the fidelity of the ruleset is at 90% for test, and 95% for training data. One possible explanation is that it is difficult to express the relationships in the images and the convolutional layers using rules. Especially the generalization to new data seems to be problematic. Rules cannot reflect image data as good as the neural network can. The NN has many more interconnections, which are difficult to be learned by the whitebox. The figures in Table 2 demonstrate that the fidelity for MNIST is the lowest among all the datasets tested. This is evidence for hypotheses 3d.

3e The extracted rules from the FF and CONV NN have the same accuracy on the test data (0.90). The accuracy for the training data is only slightly better for the CONV NN ($0.95 > 0.94$). This contradicts hypothesis 3e. Even though the accuracy of the CONV NN itself was higher than the FFNN ($0.99 > 0.96$ on test and $1.00 > 0.98$ on train), this superiority was not reflected in the extracted rules. This may be because rules generally have difficulty describing the relationships in image data.

4.1.4 Hypotheses (4) “Breast Cancer Wisconsin Dataset”

4a In the experiment, the trained network, containing of a single 60-neuron relu layer, achieved an accuracy of 0.98 (0.95) on the test (training) dataset. The rules extracted by ALPA achieved an accuracy of 0.96 (0.98) and a fidelity of 0.94 (0.99). The rules extracted by DNNRE achieved an accuracy of 0.96 (0.95) and a fidelity of 0.96 (1.0). This is evidence that, as described by hypothesis 5a, the network and rule extraction achieve a high accuracy on this dataset. The performance of the network and the rules is also slightly lower than for the Iris-dataset, however the differences between the two datasets are not large enough to make any defenit statements about that part of the hypothesis.

4b Regarding the comprehensibility, the results are again similar to the rules generated for the Iris-dataset, both for DNNRE and ALPA. This ,once again, reinforces the observation, that the pedagogical approach of ALPA comes at the cost of more rules. While DNNRE produced only 2 rules, the ALPA algorithm produced 16 rules. Regarding the complexity of the rules, the rules generated by ALPA are slightly less complex, containing on average 2.64 terms per rule for the rules for output class "M" (meaning "malign") and 1.5 for output class "B" (meaning "benign"). Whereas for DNNRE the rules for output class "M" contain on average 3.09 terms per rule and 3.55 terms per rule for output class "B". The hypothesis 5b does not hold for both of the extraction algorithms, since the generated rules are, in both cases, rather long and thus not concise. It is difficult to say if there is one set of rules, which is more comprehensible than the other. For Alpa there are a lot more rules, but for DNNRE the rules are very long overall.

4.1.5 Hypotheses (5) "UCI credit default dataset"

5a As seen in tables Table 1 and Table 2, this hypothesis remains supported. In a neural network, input values are (usually) summed up as an activation potential for a neuron. In an unencoded setting, there is no particular reason as to why an input With ID 1 should have a lower impact on the activation potential than an input with 9. Using an encoding on the other hand, both inputs would either have the same impact (One-Hot) or be weighted using their occurrence-count for each output class (WoE). The latter encoding does indeed make sense, since a higher occurring attribute value (higher WoE value) can indicate a greater impact than a lesser occurring value (lower WoE value). This allows a network/ruleset on an encoded dataset to be more precise.

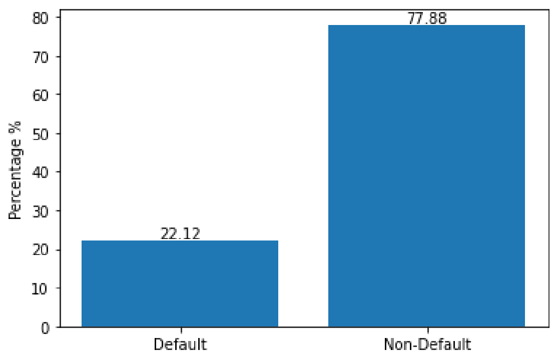


Figure 9: Percentage of occurrences of the two output classes of the UCI credit default dataset, showing the imbalance between the classes

5b As described in the section above, WoE encoding allows the different attribute values of a feature to have a differing impact on the activation potential of a neuron. This would allow a highly correlating feature (and value) to have a greater impact, allowing for more precise decisions. As seen in Table 1, this does not hold for the UCI credit default dataset. Here, the same network (besides the dimensionality of the input layer) performs better on the dataset using One-Hot instead of WoE

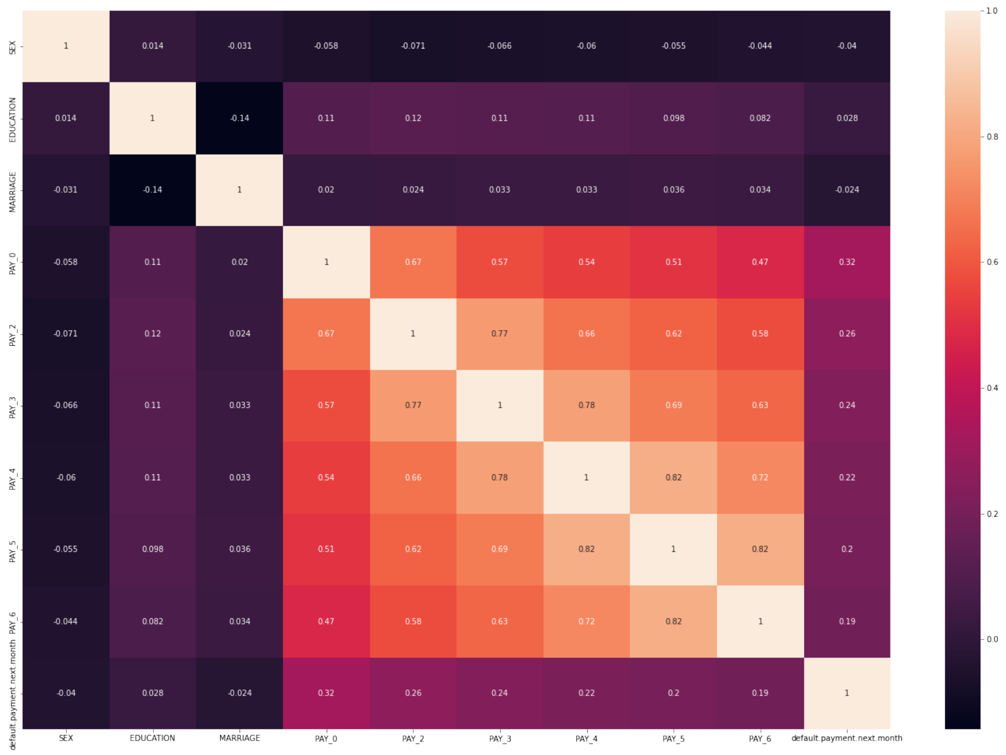


Figure 10: Correlation matrix between the different categorical features and the output classes in the UCI credit default dataset

encoding, albeit the difference is almost negligible. One can see in Figure 10 that the different categorical features are not correlating much with the output class (`default.payment.next.month`), meaning the mentioned "impact" through WoE is not that relevant. Further, looking at Figure 9, the dataset itself is very imbalanced. This can be eleviated for neural network training by weighing the output classes accordingly. However, calculation for WoE values happens on the basis of the whole dataset, meaning the calculated values are not as representative when training on a weighted set or balanced subset. This could also cause the minor accuracy difference.

5c First, one should note that rule extraction using DNNRE exceeded our time and/or memory constraints (2h and 24GB across main memory and swap space), therefore all comparisons are based on ALPA. Here, the hypothesis does also hold, albeit the fidelity difference as seen in Table 2 is again very minor. In ALPA, new datapoints are generated between datapoints close to decision boundaries between classes. For One-Hot encoded features, this means that the new datapoints are always only one of the original two, if one were to decode the conversion. With WoE however, attribute values are encoded along a number line, meaning another value could be encoded in the interval between two datapoints. Since a WoE value represents the "impact" of an attribute value in regards to an output decision, this "in-between" value therefore has its impact regarding the decision between the original two values. Coincidentally, the "impact"-value that causes the decision boundary lies between the original two values as well (since they have different classes). Therefore, the generated points with WoE encoding should be closer to the decision boundary than ones encoded with One-Hot, allowing for a more precise white box. Further, since WoE encoding leads to less input features than One-Hot encoding in this example (26 vs. 91), the whitebox can then produce better comprehensible rules (less terms per rules on average), with the caveat of more rules (79 vs 64)

4.2 Conclusion

Taking into account all the experiments and our observations while testing the implementation, we draw the following conclusions on the properties of the algorithms:

- ALPA is a fast due to its pedagogical approach, but generally produces less concise, sometimes incomprehensible, rulesets.
- DNNRE is slow, sometimes to the point of intractability, but might, in certain cases, produce more comprehensible rulesets.

This suggests that whenever extraction speed is of the essence or with limited computation power and storage, ALPA should be used and if good comprehensibility is required, DNNRE might be the right choice. We would like to note that in our experiments only a single whitebox (C5.0) was tested for ALPA. As DNNRE uses the same algorithm for rule extraction, this was a natural choice. Future works have to show whether different whiteboxes or modifications to ALPA can produce more comprehensible rulesets, combining the benefits. Different, problem targeted whitebox algorithms might also allow explanations beyond simple rules.

5 Summary

The project reached its overall goals of implementing and testing the ALPA algorithm and performing rule extraction on different neural networks. Even beyond that, rule extraction from convolutional networks was carried out.

During the project, one what-if-scenario had to be executed: Due to a prolonged period of severe illness, the team member Mahla Haghzad had to leave the project. As specified, the workload was redistributed.

Future work might include exploring better ways of extracting rules from convolutional networks, as well as exploring different extraction techniques. Also, different whitebox models need to be tested with the ALPA algorithm. A lot of time is spent (brute-force) optimizing the rho-parameter of the ALPA algorithm. Some studies on which rho-values might be a good choice for a given network/dataset could make the algorithm a lot faster in the future.

A Appendix

A.1 Search Queries

The following list details the search queries executed on each search engine. For some of the search engines and queries, the custom search options were used.

- Google Scholar (`https://scholar.google.de/`)
  - `"rule extraction" AND "sampling" AND "neutal network"`
  - `"pedagogical" AND "rule extraction"`
  - `fuzzy rule extraction neural networks`
  - `black box rule extraction neural network`
  - `validity interval analysis neural networks`
  - `reverse engineering pedagogical neural network`
  - `"reverse engineering" AND "neural networks" AND "rule extraction"`
- Research Gate (`https://www.researchgate.net/`)
  - `"rule extraction" AND "sampling"`
  - `fuzzy rule extraction neural networks`
  - `black box rule extraction neural network`
  - `validity interval analysis neural networks`
  - `reverse engineering rule extraction neural network`
- ACM Digital Library (`https://dl.acm.org/`)
  - `"[All: rule extraction] AND [All: sampling] AND [All: neural network]"`
  - `"[Title: "rule extraction"] AND [Abstract: "neural network"] AND [Abstract: sampling]"`
  - `fuzzy rule extraction neural networks`
  - `black box rule extraction neural network`
  - `validity interval analysis neural networks`
  - `All: reverse engineering rule extraction neural network`
  - `"[All: all: reverse engineering] AND [All: neural network]"`
- DBLP (`https://dblp.org/`)
  - `fuzzy rule extraction neural networks`
  - `validity interval analysis neural networks`
- IEEE Xplore (`https://ieeexplore.ieee.org`)
  - `("All Metadata":rule extraction) AND ("All Metadata":neural network) AND ("All Metadata":sampling)`

  - `fuzzy rule extraction neural networks`
  - `black box rule extraction neural network`
  - `validity interval analysis neural networks`
  - `reverse engineering neural network`
  - `rule extraction, pedagogical approach, neural networks`
  - `pedagogical approach, rule extraction`
- Springer Link (`https://link.springer.com/`)
  - `sampling AND "rule extraction" AND (neural)`
  - `"fuzzy rule extraction" AND "from neural networks" [no books]`
  - `black box rule extraction neural network AND article`
  - `black box rule extraction neural network AND conference paper`
  - `reverse engineering rule extraction neural network`
  - `reverse engineering pedagogical neural network rule extraction`
- Elsevier Science Direct (`https://www.sciencedirect.com/`)
  - `"rule extraction" AND "neural network" + keywords:  sampling`
  - `fuzzy rule extraction neural networks`
  - `black box rule extraction neural network`
  - `validity interval analysis neural networks`
  - `reverse engineering+"neural network"+ "rule extraction"`

A.2 Evaluation Criteria

- pedagogical approach (y/n)
  - whether the paper proposes an approach for extracting rules from ANN without layer decomposition
- electic approach (y/n)
  - whether the paper proposes an approach for extraction rules without complete layer decomposition, but some information about the network structure
- name of the algorithm
  - the name, which the authors gave their algorithm
- maximum number of hidden layers
  - if the paper has anything to say about this and whether the technology is suitable for DNNs
- network type agnostic (y/n)

  - whether the proposed approach can be used on a variety of networks (e.g. not CNN- specific)

- type of ANN if not type agnostic
- type of explanations (e.g. Rules/Decision Trees, no saliency maps)
  - the type of explanations in the output of the proposed approach
- experimental evaluation (y/n)
  - whether the paper applies the proposed method to an ANN and evaluates the experiment
- results
  - machine learning domain (classification, regression, ...)
  - final accuracy (how well the approach worked, e.g. fidelity score)
  - model/problem used to evaluate accuracy (e.g. XOR-Problem)
- degree of reproducibility
  - list of aspects of the paper's content which benefit its reproducibility or just the best of those aspects
  - formulas, pseudo- code, code, high- quality code (high-level language, maintained repository
- code url if there is one
- understandable content (rating 1-5)
  - rating of the content in terms of subjective understandability from 1 to 5
- additional notes
  - special aspects of the paper which seem to be important
  - e.g. a special purpose of the extracted rules or an important coupled paper

A.3 Literature Evaluation Table

topic	metadata			approach							evaluation(s)				overall quality		
criterion	id	title	url	pedagogical	eclectic	algorithm_name	max_number_of_hidden_layers	type_agnostic		rule_type(s)	has_evaluation	ml_domain	model_or_problem	results	reproducibility	code_url	understandable
datatype	integer	string	url	yes/no	yes/no	string	integer	true/false	if FALSE: type of ANN	if-then, fuzzy, decision_tree, no_saliency, ...	yes/no	classification, regression, ...	XOR, Iris, ...	accuracy x.y%, fidelity x.y	formulas, pseudo-code, co	url/0	1-5, 1<2<3<4<5
description	The number of the entry	The title of the paper	The url leading to the paper, preferrably a doi	whether the approach works **without layer decomposition**	the approach needs **limited** information about the network	the name the authors gave their algorithm	the maximum number of **hidden layers** (if not a deep network 1, if not known ?)	whether the algorithm is **independent** of the type of network used		the type of explanations in the output of the proposed approach	whether the algorithm is evaluated	the machine learning domain the algorithm is applied to	the model the evaluation is done with	the final score on the given domain and model	a list of things the paper contains	the url to the code accompanying the paper or 0 if there is none	how understandable the content is from 1 to 5
	1	a generic fuzzy aggregation operator: rules extraction from and insertion into artificial neural networks	https://doi.org/10.1007/s00500-007-0221-8	no	no												1
	2	a modified fuzzy min-max neural network with rule extraction and its application to fault detection and classification	https://doi.org/10.1016/j.asoc.2007.07.013	no	yes	fuzzy min-max nn	1	FALSE	Fuzzy Min-Max	fuzzy	yes	classification	PID, WBC	accuracy 74.28%, 96.42%	formula		2
	3	extracting fuzzy rules from hierarchical heterogeneous neural networks for cardiovascular diseases diagnosis	https://doi.org/10.1007/978-3-642-53917-6_22	yes	yes	genetic algorithm for hierarchical heterogen	?	FALSE	hierarchical heterogeneous	fuzzy	yes	classification	CVD		flow chart		3
	4	extraction of similarity based fuzzy rules from artificial neural networks	https://doi.org/10.1016/j.ijar.2006.04.003	no	no												
	5	fuzzy difaconn-miner: a novel approach for fuzzy rule extraction from neural networks	https://doi.org/10.1016/j.eswa.2012.05.050	no	yes	fuzzy DIFACONN-miner	?	FALSE	Feed Forward	fuzzy	yes	classification	Bupa, Cleveland, Gl	66% to 97% in testing	flow chart, pseudo-code		3
	6	fuzzy logic and evolutionary algorithmâ€"two techniques in rule extraction from neural networks	https://doi.org/10.1016/j.neucom.2004.04.015	yes	no	REX Pitt, REX Michigan (genetic algorithms	?	TRUE		fuzzy	yes	classification	Iris, WINE, WBCD	accuracy 97.33%, 93.73%	flow chart, test-procedure		4
	7	knowledge extraction from neural networks using the all-permutations fuzzy rule base: the led display recognition problem	https://doi.org/10.1109/TNN.2007.891686	no	no												
	8	Application of a Neural Fuzzy System with Rule Extraction to Fault Detection and Diagnosis	https://doi.org/10.1007/s10845-005-4371-1	no	yes	fuzzy min-max nn	1	FALSE	Fuzzy Min-Max								1
	9	cracking the black box: distilling deep sports analytics	https://doi.org/10.1145/3394486.3403367	?	yes	mimic learning with tree models		FALSE	sports analytics	decision_tree	yes	regression	sports analyytics				3
	10	eclectic extraction of propositional rules from neural networks	https://doi.org/10.1109/ICCITechn.2011.6164790	no	yes	HERETIC		FALSE	feed forward, MLP, only sigmoid activ	decision_tree, if-then	yes	classification	Promoters, Breast C	accuracy from 85 to 97 %, f	pseudo-code		4
	11	explaining black box models by means of local rules	https://doi.org/10.1145/3297280.3297328	yes	no	LACE	?	TRUE		if-then	yes	classification	Monk, Adult				2
	12	function analysis based rule extraction from artificial neural networks for transformer incipient fault diagnosis	https://doi.org/10.1016/j.ijepes.2012.06.042	yes	no	REFANNs	1	FALSE	Feed Forward	if-then with molecule gas analysis	yes	regression	transformer fault diagnosis				3
	13	mining classification rules from fuzzy min-max neural network	https://doi.org/10.1109/ICCCNT.2014.6963079	no	yes		1	FALSE	Fuzzy Min-Max	if-then	yes	classification	Iris, Wine	accuracy depends on parameter delta			2
	14	neural data analysis: ensemble neural network rule extraction approach and its theoretical and historical backgrounds	https://doi.org/10.1007/978-3-642-38658-9_1	no	yes	Ensemble-Recursive-Rule eX	1	TRUE	back-propagation / Ensemble	if-then	yes	classification	CARD3, German Cr	accuracy from 79% to 99%	pseudo-code		4
	15	patient admission prediction using a pruned fuzzy min–max neural network with rule extraction	https://doi.org/10.1007/s00521-014-1631-z	yes	no	fuzzy min-max nn	1	FALSE	Fuzzy Min-Max	if-then	yes	classification	Iris, PID, Sonar	56.09 to 95.75%			4
	16	regression rules extraction from artificial neural network based on least squares	https://doi.org/10.1109/ICNC.2011.6021906	yes	no	least squares	?	FALSE	Feed Forward	if-then	yes	regression	Computer Hardware	MAE: 7.866, RMAE: 0.089	formulas		3
	17	research on rules extraction from neural network based on linear insertion	https://doi.org/10.1109/ICIE.2010.103	yes	no	Linear Insertion	?	FALSE	Feed Forward	decision_tree	yes	regression	Concrete Compress	MAE: 6.438, RMAE: 0.209	formulas		3
	18	rule extraction from an optimized neural network for traffic crash frequency modeling	https://doi.org/10.1016/j.aap.2016.08.017	no	yes	Approximating transfer functions	?	TRUE		transfer functions	no	regression					4
	19	rule extraction from constructively trained neural networks based on genetic algorithms	https://doi.org/10.1016/j.neucom.2011.04.009	yes			1	FALSE	Feed Forward trained with constructiv	if-then, oblique rules	yes	classification	Monk, Breast Cance	accuracy 95%-100%	formulas		5
	20	rules extraction in svm and neural network classifiers of atrial fibrillation patients with matched wavelets as feature generator	https://doi.org/10.1109/IEMBS.2009.5334220	yes	no	TREPAN	?	TRUE		if-then	no	classification					5
	21	Accuracy of rule extraction using a recursive-rule extraction algorithm with continuous attributes combined with a sampling tec	https://doi.org/10.1016/j.imu.2016.10.001	yes	no	Sampling-Continuous Re-RX	study used up to 3 but claims it w	FALSE	Sampling step creates an Ensemble c	decision_tree, if-then	yes	classification	BUPA Liver Disorder	CV, acc 73%	pseudo-code, diagrams, ta	0	1
	22	Active Learning-Based Pedagogical Rule Extraction	https://doi.org/10.1109/TNNLS.2015.2389037	yes	no	ALPA	positive_infinity	TRUE	-	any	yes	classification	many	numbers given, but difficult	formulas, pseudo-code, tal	(java) https://www	5
	23	A new itj method with combined sample selection technique to predict the diabetes mellitus	https://doi.org/10.26483/ijarcs.v8i9.4914														
	24	A Neural Data Analysis Approach Using Ensemble Network Rule Extraction	https://doi.org/10.1007/978-3-642-33269-2_65	no	yes	E-Re-RX	study uses only 1	FALSE	Ensemble Neural Networks								
	25	Exact and Approximate Rule Extraction from Neural Networks with Boolean Features	https://doi.org/10.5220/0008362904240433	yes	no	thruth-table sampling	1 with 10 hidden units, theoretical	FALSE	Boolean output only	boolean expressions	yes	boolean classification	cross-site scripting	accuracy 99.9%	pseudo-code, tables		5
	26	Extracting Rules from Neural Networks as Decision Diagrams	https://doi.org/10.1109/TNN.2011.2106163	no	yes	LORE (LOcal Rule Extraction)	positive_infinity	FALSE	MLP with discrete inputs	decision diagram, boolean expressions	yes	boolean classification	MONKs	max 7.1% errors	formulas, pseudo-code, tables		4
	26												UCI dataset	max 0.03% errors on krkpa7			
	27	Genetic Rule Extraction Optimizing Brier Score	https://doi.org/10.1145/1830483.1830668	yes	no	G-REX	positive_infinity	TRUE	-	any	yes	classification	Weka	mean accuracy 78%	formulas, pseudo-code (in	https://sites.goog	4
	28	Neural Network Rule Extraction to Detect Credit Card Fraud	https://doi.org/10.1007/978-3-642-23957-1_12	yes	no	SOAR II (Sparse Oracle-based Adaptive Rule extration)		TRUE	-	if-then	yes	classification	Credit Card Fraud	accuracy 93%, fp-rate 1:21	formulas, tables		5
	29	Reverse Engineering the Neural Networks for Rule Extraction in Classification Problems	https://doi.org/10.1007/s11063-011-9207-8	no	yes	RxREN (Rule Extraction by REverse Engine	positive_infinity	FALSE	Feed Forward	if-then-else	yes	classification	iris	accuracy 97.3%	formulas, diagrams, pseudo-code, tables		3
	29												creditg	accuracy 72.2%			3
	29												iono	accuracy 94.3%			
	30	Rule Extraction from Neural Network Using Input Data Ranges Recursively	https://doi.org/10.1007/s00354-018-0048-0	no	yes	ERENNR (Eclectic Rule Extraction from Ne	1 used	FALSE	Feed Forward	if-then	yes	classification	credit approval	accuracy 93.89%, fidelity 9	formulas, pseudo-code, tables		4
	31	Rule Extraction from Neural Network trained using deep belief network and back propagation	https://doi.org/10.1007/s10115-020-01473-0	no	yes	ERENN_MHL (Eclectic Rule Extraction fron	more than 1	FALSE	Feed Forward	if-then	yes	classification	credit approval	accuracy 86.26%	formulas, pseudo-code, diagrams, tables		4
	32	Time-to-event analysis with artificial neural networks: An integrated analytical and rule-based study for breast cancer	https://doi.org/10.1016/j.neunet.2007.12.034	no	no	Only Application of Orthogonal Search-based Rule Extraction (OSRE), not	1	FALSE	MLP								
	33	Orthogonal Search-Based Rule Extraction (OSRE) for Trained Neural Networks: A Practical and Efficient Approach	https://doi.org/10.1109/TNN.2005.863472	Yes	No	OSRE	positive_infinity	FALSE	Boolean output	boolean expressions	yes	classification	MONKs	accuracy 94.8%	formulas, pseudo-code, tables		4
	33												Wisconsin Breast Canser				
	34	Towards Reverse-Engineering Black-Box Neural Networks	https://doi.org/10.1007/978-3-030-28954-6_7	Yes	No	kennen	1	FALSE	Feed Forward	if-then	yes	classification	MNIST	accuracy71.8% and 90.0%	formulas,diagrams, tables	https://github.con	3
	35	Rule extraction from the Artificial Neural Network	https://doi.org/10.1088/1757-899X/1029/1/012127	No	No												
	36	Reverse-Engineering Deep Neural Networks Using Floating-Point Timing Side-Channels	https://doi.org/10.1109/DAC18072.2020.9218707	No	No												
	37	Reverse-engineering bar charts using neural networks	https://doi.org/10.1007/s12650-020-00702-6	No	No												
	38	Knowledge discovery in corporate events by neural network rule extraction	https://doi.org/10.1007/s10489-007-0053-3			REFANN											
	39	Logic mining in neural network: reverse analysis method	https://doi.org/10.1007/s00607-010-0117-9	No	?	Reverse Analysis	?	FALSE	recurrent Little-Hopfield			classification			formulas		2
	40	Extracting Classification Rules from Artificial Neural Network Trained with Discretized Inputs	https://doi.org/10.1007/s11063-020-10357-x	No	Yes	MNNGR	?	TRUE		if-then	yes	classification	many	Numbers are given for diffe	formulas, tables		4
	41	SNIFF: Reverse Engineering of Neural Networks With Fault Attacks	https://doi.org/10.1109/TR.2021.3105697	Yes	Yes	SNIFF-sign bit flip fault	1	FALSE	Feed Forward	if-then	yes	classification			formulas		3
	42	reliable estimation of a neural network's domain of validity through interval analysis based inversion	https://doi.org/10.1109/IJCNN.2015.7280794	Yes	No	SIVIA	?	FALSE	Feed Forward	domain of validity	yes	classification	XOR, Ring-Spot and more		formulas, pseudo code		2
	43	reachable sets bounding for generalized neural networks with interval time-varying delay and bounded disturbances	https://doi.org/10.1007/s00521-016-2580-5	No	No	no algorithm, only formulas	?	FALSE	generalized neural networks with interval time-varying delay and bounded disturban		no, but "numerical examples"				formulas		1
	44	divide and slide: layer-wise refinement for output range analysis of deep neural networks	https://doi.org/10.1109/TCAD.2020.3013071	No	No	mixed-integer linear programming, sliding w	10	TRUE		output range	yes	classification	MNIST, CIFAR		formulas, pseudo code		3
	45	Neural network explanation using inversion	https://doi.org/10.1016/j.neunet.2006.07.005	yes	no	HYPINV	positive_infinity	FALSE	binary output only	rules in the form of conjunction and disjunctio hyperplanes	yes	classification	XOR and many more		pseudo-code, python imple	https://github.con	3
	46	X-TREPAN: a multiclass regression and adapted extraction of comprehensible decision tree in artificial neural networks	https://doi.org/10.5121/csit.2015.51405	yes	no	X-TREPAN	positive_infinity	TRUE		decision_tree	yes	classification	Body Fat	accuracy 94%	pseudo-code, python imple	https://github.con	5

A.4 User Documentation

This document describes the installation and usage of the Rule Extraction Assistant (REA).

Required Software

This project is based on Python and requires `python3.9`.

In order to install and run REA, the following python packages are needed: - `pandas` - `numpy` - `scikit-learn` - `tensorflow` - `rpy2`

They can be installed using `pip`, `venv`, `virtualenv`, etc. and the provided `Pifile` or `requirements.txt`.

The following non-python software is needed for the rule extraction with ALPA or DNNRE: - `R` (programming language/interpreter) - R `C50` package

Example Setup

- Install python using your Linux distributions package manager or from the official website
- Install R with the C50 library
  - intsall `R` using your Linux distributions package manager or download it from the official website
  - to install `C5.0`, invoke the `R` REPL on a command prompt (root privileges might be necessary to write to `usr/lib`)
  - type `install.packages("C50")`
  - for this, you may need `build-essentials` (debian) or at least `gcc` and `gfortran`
- Set up the virtual environment
  - run `pipenv install --python 3.9` in the source folder

Usage

You can use the tool either through its API or through its CLI.

API

The api is documented in the API Documentation, which is also contained in the `docs` folder of the implementation.

CLI

We provide a CLI, which accepts a list of configuration files and executes the specified pipeline run. Different runs can be achieved by executing the CLI multiple times with different configuration file(s). The generated output files of each module can then also be used by other programs (provided that they can read the format). Some (advanced) examples for the usage of the CLI can be found in the experiments folder and the `run.sh` scripts in the hypothesis folders.

The next section provides you with an overview of all the configuration parameters, their data-type, restrictions and also whether they are required.

Execute the CLI by running `python -m rea -h` (in the source folder or after installing). This will provide you with some help.

Configuration Format

Global Keys

Key name	Name in Code	optional/required	description
logging	GlobalKeys.LOGGING	optional	output verbosity. One of the python logging modules log levels
seed	GlobalKeys.SEED	optional	seed used by all modules for reproducibility
metrics_filename	GlobalKeys.METRICS_FILENAME	optional	filename for the metrics generated by the extraction module
rules_filename	GlobalKeys.RULES_FILENAME	optional	filename for the rules generated by the extraction module

Key name	Name in Code	optional/required	description
predict_instance_filename	PREDICT_INSTANCE_FILENAME	optional	name of the output pickle instance, is being used in extraction and evaluation

Keys and Outputs of the Data-Module

Key name	Name in Code	optional/required	description
input_path	DataKeys.INPUT_PATH	optional	path to the dataset from which rules are to be extracted
output_path	DataKeys.OUTPUT_PATH	optional	Path to the folder to fill with output. If directory doesn't exist, one will be created.
label_col	DataKeys.LABEL_COL	optional	index or name of the column containing the labels, i.e. the feature to be predicted
orig_shape	DataKeys.ORIG_SHAPE	optional	specifies the original shape of the dataset before any conversion
test_size	DataKeys.TEST_SIZE	optional	percentage of the data used for testing
dataset_name	DataKeys.DATASET_NAME	optional	friendly name to give to the dataset
cat_conv_method	DataKeys.CAT_CONV_METHOD	optional	Method for categorical conversion
categorical_columns	DataKeys.CATEGORICAL_COLUMNS	optional	List of categorical columns to be converted in the dataset
scale_data	DataKeys.SCALE_DATA	optional	flag to MinMaxScale the input data

File	Description
`x_train.npy`	Training data in numpy format
`y_train.npy`	Trainin labels in numpy format
`x_test.npy`	Test data in numpy format
`y_test.npy`	Test labels in numpy format
`encoder.pickle`	Lable encoder, instance of the scikit `LabelEncoder` serialized with pickle
`metadata.json`	Remaining important fields, like `original_size`

Keys and Output for the Model-Module

Key name	Name in Code	optional/required	description
nwtype	ModelKeys.TYPE	required	The type of network to use ("ff" or "conv")
hidden_layer_units	ModelKeys.HIDDEN_LAYERS	required	the number layers and number of units for each hidden layer
hidden_layer_activations	ModelKeys.HIDDEN_LAYER_ACTIVATIONS	required	specifies the activation function used for each hidden layer
conv_layer_kernels	ModelKeys.CONV_LAYER_KERNELS	required	kernel to be used in each layer of a convolutional network
use_class_weights	ModelKeys.USE_CLASS_WEIGHTS	optional	flag that enables or disables precomputed class_weights
batch_size	ModelKeys.BATCH_SIZE	optional	number of samples per gradient update
epochs	ModelKeys.EPOCHS	optional	number of epochs to train the model
output_path	ModelKeys.OUTPUT_PATH	required	path where to save the model

Key name	Name in Code	optional/required	description
learning_rate	ModelKeys.LEARNING_RATE	optional	Learning rate for adam optimizer or initial learning rate for exponential decay
use_decay	ModelKeys.USE_DECAY	optional	flag that enables use of adam with exponential decay
dropout	ModelKeys.DROPOUT	optional	Rate for keras `Dropout` layer
val_split	ModelKeys.VAL_SPLIT	optional	percentage of the training data used for validation
data_path	ModelKeys.DATA_PATH	required	path to the `Data` output folder

File	Description
`history.png`	Visualization of training-process measures (validation loss/accuracy)
Keras Model Files	The trained tensorflow model

Keys and Output for the Extraction-Module

Key name	Name in Code	optional/required	description
trained_model_path	ExtractionKeys.MODEL_PATH	required	path where to load the model from
data_path	ExtractionKeys.DATA_PATH	required	path to the `Data` output folder
algorithm	ExtractionKeys.ALGORITHM	required	extraction algorithm to use, either “alpa” or “dnnre”
rules_dir	ExtractionKeys.OUTPUT_PATH	required	path of folder to save rules and metrics

File	Description
`eval_metrics.json`	Contains metrics, such as execution time, memory and best rho for ALPA
`rule_classifier.pickle`	R C5.0 prediction instance serialized with pickle
`rules.bin`	Extracted rules, serialized with pickle from the internal format

Keys and Output for the Evaluation-Module

Key name	Name in Code	optional/required	description
trained_model_path	EvaluationKeys.MODEL_PATH	required	path to the trained tensorflow model
rules_dir	EvaluationKeys.RULES_DIR	required	path to the rules folder
data_path	EvaluationKeys.DATA_PATH	required	path to the `Data` output folder
evaluation_dir	EvaluationKeys.OUTPUT_PATH	required	path of folder to save results of evaluation

File	Description
`test_eval.json`	Raw data produced by the evaluation on test set
`test_eval.md`	Evaluation report (based on raw data) for test set
`train_eval.json`	Raw data produced by the evaluation on train set
`train_eval.md`	Evaluation report (based on raw data) for train set
confusion matrices	pngs of the confusion matrices

A.5 Developer Documentation

This is the developer documentation of the Rule Extraction Assistant (REA), contained in this repository. It includes: - What are the modules? - How do they interact? - Which software patterns have been used? - What software-design choices have been made - and why?

Project Structure

- the source code can be found in `40_Realisation/99_Final_System`
- The content of the folder is structured as follows:

Root

Name	Purpose
`Pipfile`	Pipfile to be used with pipenv to create the venv necessary for development
`requirements.txt`	List of required python packages. Can be used with the venv module
`.pre-commit-config.yaml`	Configuration for pre-commit hooks

Experiments folder

Name	Purpose
`experiments`	All experiments/studies conducted using the rea tool
`experiments/datasets`	Datasets used for experiments

Units Tests

Name	Purpose
`test`	Unit tests for the rea python module
`test/resources`	some resources (data, configuration files, …) used for the unit tests

REA Program

Name	Purpose
`rea`	The python code of rea
`rea/rea.py`	Main python class and cli
`rea/configuration.py`	Json configuration reader/validator
`rea/data`	Data loading and preprocessing module
`rea/model`	Neural network training module
`rea/extraction`	Rule extraction module
`rea/extraction/alpa`	ALPA rule extraction algorithm
`rea/extraction/dnnre`	DNNRE rule extraction algorithm (implemented by sumaiyah)
`rea/rules`	DNF rule representation in python (implemented by sumaiyah)
`rea/evaluation`	Module and functions for the evaluation of extracted rules (fidelity, comprehensibility, accuracy, …)

Dependencies

For the execution/building of the project, `Python 3.9` is required.

The following python packages are dependencies of rea: `- pandas - numpy - scikit-learn - category_encoders - tensorflow - rpy2 - Jinja2`

The following non-python software is needed: `- R` (programming language/interpreter) - R `C50` package

The following are dependencies only necessary for development: `- pre-commit - flake8 - jupyter`

3rd Party Code

The code for rea was based on a previous NEidI project implementation of Dnnre. However, except for the general pipeline structure and style of configuration, there is not much left from this implementation.

The code for dnnre, the rule representation and the rule evaluation is taken from sumaiyah. Some modifications were made, especially the evaluation code was heavily modified. The files were also moved to the proper modules. The documentation of the modifications can be found at the top of the respective files. We note that the repository of sumaiyah has no license attached, which prohibits the use of his code for non-private purposes in a legal sense. If this project was to be made public, the affected part would need to be replaced or the aforementioned author contacted to provide an (open-source) license (like Apache-2.0).

Software Design

REA follows a modular structure, where each module works independently to produce certain output files from given input files (which usually are ouputs of other modules), forming a pipeline-like data-flow. The pipleine can be configured using a file in `json` format following a certain structure/schema.

This will explain the software design, for information on how to use the modules/configuration, refer to the user documentation.

Pipeline Structure

The following modules are available: `- data - model - extraction - evaluation`

The **data** module is the foundation of the pipeline. It will load a provided dataset into a pandas dataframe and apply pre-processing steps, such as one-hot encoding for labels and weight of evidence encoding for categorical features. At the moment, only csv(`.csv`) or hdf5 (`.hdf5`, `.h5`) formats are supported. It is however easy to add support for another pandas-supported format by modifying the loading procedure in `data.py`.

The **model** module uses `tensorflows` `keras` API to construct and train feed-forward or convolutional neural networks on a provided dataset. The constructed neural networks are general purpose and thus the ability to tailor them to a specific problem is limited. Basic parameters, like the number of hidden layers, kernel size, but also the learning rate, exponential decay confifguration and also dropout factor are exposed in the configuration file. For cases that need very well constructed and trained networks, it is recommended to supply a pre-trained model to the pipeline. This module can however be used for experimenting with the tool, wehere automatic creation of networks is advantageous.

The **extraction** module uses either the dnnre or alpa algorithm to extract rules from a trained tensorflow model and the provided data. It also collects some metrics, for example the execution time and memory consumption, on this process for later evaluation.

The **evaluation** module calculates some metrics on generated rules, the neural network and the extraction algorithm metrics and compiles them into markdown reports for the training and test datasets respectively.

Pipeline Configuration

To describe a (reproducible) pipeline run, configuration file(s) need to be supplied for each run. The minimal configuration accepted is the global section with the seed and logging level. A minimal *useful* configuration additonally contains at least of invoking the data module and possibly one of the three other modules. Each module has its own section in the configuration file, which follows the json format. In each section, the input and output paths are specified. The contents of the configuration file can be split into multiple files, allowing the combination and re-use of different module configurations. For example, one might want to only execute the data module once and then train two different networks on the pre-processed dataset.

API vs CLI

We provide a CLI, which accepts a list of configuration files and executes the specified pipeline run. Different runs can be achieved by executing the CLI multiple times with different configuration file(s). The generated output files of each module can then also be used by other programs (provided that they can read the format). Some (advanced) examples for the usage of the CLI can be found in the experiments folder.

Additionally, it is also possible to use the tool through its API by importing the rea module in other python projects. It is also possible to replace existing modules with custom implementations by inheriting from on of the module superclasses.

Development

Prerequisites

- Python 3.9
- R with the C50 library
  - to install, invoke the `R` REPL (root privileges might be necessary to write to `usr/lib`)
  - type `install.packages("C50")`
  - for this, you may need `build-essentials` (debian) or `gcc` and `gfortran`

Contributing

1. create a virtual python environment, for example `pipenv install --python 3.9`
2. activate the environment, for example `pipenv shell`
3. run `pre-commit install` to add the commit hooks to your git hooks
4. run `pre-commit run --all-files` once to apply all hooks for the first time
5. develop, develop, develop
6. stage and commit your changes; if you get an error during pre-commit, you need to stage the changes by pre-commit and commit again
7. push to the remote
8. goto (4)

Tests

Unit tests for the source code are provided in the `tests` folder. You can use the `run_all.sh` script to run them all. We use the standard `unittest` API provided by Python.

API Documentation

Docstrings are provided in the source code, so you can use `pydoc` or the tools of your IDE to read their documentation. Beyond that, for developers or other very interested individuals, a lot of source-code comments are provided for potentially unclear/complex sections.

In-Depth Information

`data` Folder

The `Data` class implements the preprocessing. Modules which use `Data` for data preprocessing inherit the `ProcessingModule` base class. Data uses `scikit-learn` to split input data into test and training sets and `sklearn.preprocessing.LabelEncoder` for one hot encoding of the class labels. To handle nominal or ordinal input data we added an interface to scikit-learns `category_encoders` package. This allows the use of numerous encodings, such as One Hot, Helmert, Leave One Out or ordinal. More encoding functions can be easily added (dictionary `Data.cat_encoder_methods`). As default encoding method for nominal attributes we implemented a custom, Numpy-based version of Weight Of Evidence encoding which can even handle non-binary classes. The implementation follows *Transformation of nominal features into numeric in supervised multi-class problems based on the weight of evidence parameter* (http://dx.doi.org/10.15439/2015F90).

`model` Folder

The `Model`class uses Tensorflow and Keras to generate an artificial neural network. There is support for Feed-Forward and Convolutional networks with associated parameters. The module adds a `keras.layers.Dropout` layer before the output layer per default to avoid overfitting. Convolutional networks contain of a series of `Conv2D` and `MaxPool2D(2, 2)` layers. Some learning parameters (learning rate, exponential decay, . . .) are also exposed for configuration. Generally, the default parameters are chose to support common use-cases and need to be adapted using the configuration if necessary. When training the model, a callback is used to always only store (and potentially overwrite) the most accurate model in terms of validation accuracy. For this, a validation split is made on the fly by using the `validation_split` property of the `model.fit` function. Finally, some reports over the training history are generated to support an iterative training process.

`extraction` Folder

The `Extraction` class wraps rule extraction methods. It assures that time and memory usage of the algorithms are recorded in a comparable manner (`Extraction.metrics`). We support DNNRE and ALPA. More algorithms can be added by a corresponding folder and another `Extraction` instance method. The present rule representation in the `rules` folder should be used for a comparable evaluation. Memory measurements might exhibit some strange behaviour that could not be explained. Most likely, this occurs when using a GPU with tensorflow.

`DNNRE` The DNNRE implementation was completely taken over from the previous group, which in turn took over the code from sumaiyah. This is why the current DNNRE code only supports Feed-Forward neural networks.

`ALPA` Our Alpa implementation grew out of a review of the Java code for Weka . The program flow in the loop of the `alpa` method as well as the auxiliary methods are based on it. We eliminated some inefficiencies and peculiarities of the original implementation (like redundant network predictions). Since a pure Python implementation would produce a high computational overhead, we decided to implement the computationally intensive operations using the Numpy library. This was especially necessary for the nearest neighbor calculation in `Alpa.get_nearest` to match the quadratic effort. As a whitebox, we used the C5.0 decision tree, since it is also used in DNNRE and is considered one of the best. We use the same R interface to train the decision tree and generate rules from it. The `R` interface in turn uses a single threaded C implementation, which is freely available under the GNU General Public License. The computational effort within the Alpa loop mainly originates from training, and classifying with, the whitebox instances. To make the best whitebox instance reusable in the `Evaluation`, we save it with `pickle`. Other whiteboxes could be added analogously to `alpa_c5.py`. A function for model generation and a function for classifying new data would have to be implemented.

`rules` Folder

This folder contains the parts of the code of the sumaiyah DNNRE implementation which we use for a common basis of evaluation and rule extraction. The intention is to create rules with different extraction algorithms in the same format, so that the evaluation is not dependent on the method used.

`evaluation` Folder

The `evaluate_rules` folder contains the parts of the code of the sumaiyah DNNRE implementation which we use for a common evaluation of the extracted rules. We have made some parts more efficient by using Numpy operations and fixed bugs. Furthermore, we added functions for pretty printing of rules and attribute counting for evaluation purposes. The latter is especially intended for creating heatmaps when evaluating image data. The DNNRE `predict` function in `evalue_rules/predict.py` has a high Python overhead and takes a long time to predict high dimensional data such as images. This is why we added the possibility of using a prediction instance in `Evaluation.load_predict_instance`. We advise to use this feature for evaluation rules that are extracted with the alpa algorithm. For instance, only with the help of the `R C5.0` prediction instance we were able to predict the MNIST dataset with rules in reasonable time. The `Evaluation` class uses a `Jinja2` template to generate Markdown files with the evaluation output. The template is stored in `template/eval_templ.md`.

`configuration.py`

This file contains classes for easy access to and renaming of dictionary keys. This is useful for autocompletion and also for preventing KeyErrors. The `Configuration` class has validation methods for each of the REA modules. They are intended to be used before the modules are run. This is to catch as many usage errors as possible before the pipeline is actually executed.

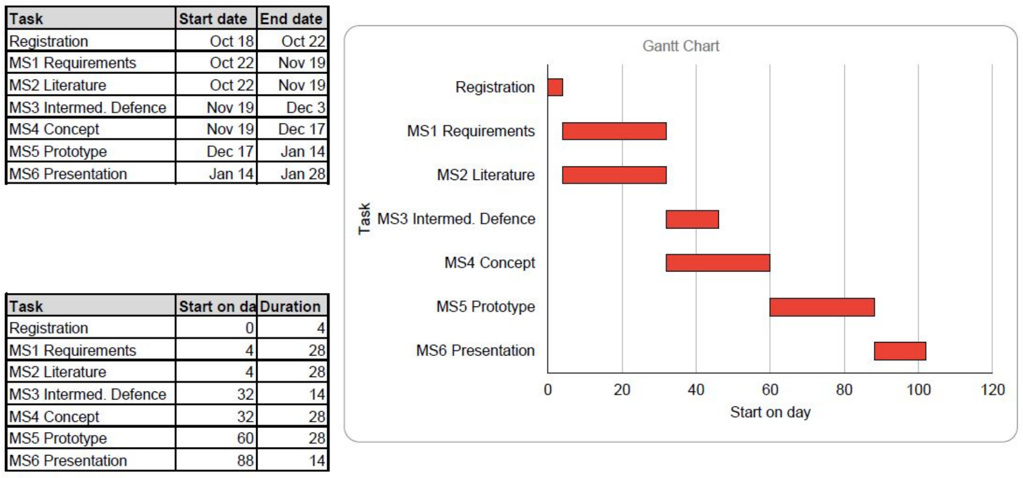
`rea.py`

The `REA` class combines the modules into the pipeline. It also implements the two user interfaces (Cli and API). The CLI flags are generated in `__main__.py`, so that the `rea` module can be used with `python -m rea` command.

Pre-Commit

Pre-Commit is used for consistent collaboration. It ensures a unified code-style and enforces other constraints, like a filesize limit.

A.6 GANTT Project Visualisation



Task	Start date	End date
Registration	Oct 18	Oct 22
MS1 Requirements	Oct 22	Nov 19
MS2 Literature	Oct 22	Nov 19
MS3 Intermed. Defence	Nov 19	Dec 3
MS4 Concept	Nov 19	Dec 17
MS5 Prototype	Dec 17	Jan 14
MS6 Presentation	Jan 14	Jan 28

Task	Start on da	Duration
Registration	0	4
MS1 Requirements	4	28
MS2 Literature	4	28
MS3 Intermed. Defence	32	14
MS4 Concept	32	28
MS5 Prototype	60	28
MS6 Presentation	88	14

Figure 11: Visualisation of our project as Gantt Chart.

A.7 Timesheet Pie Plots

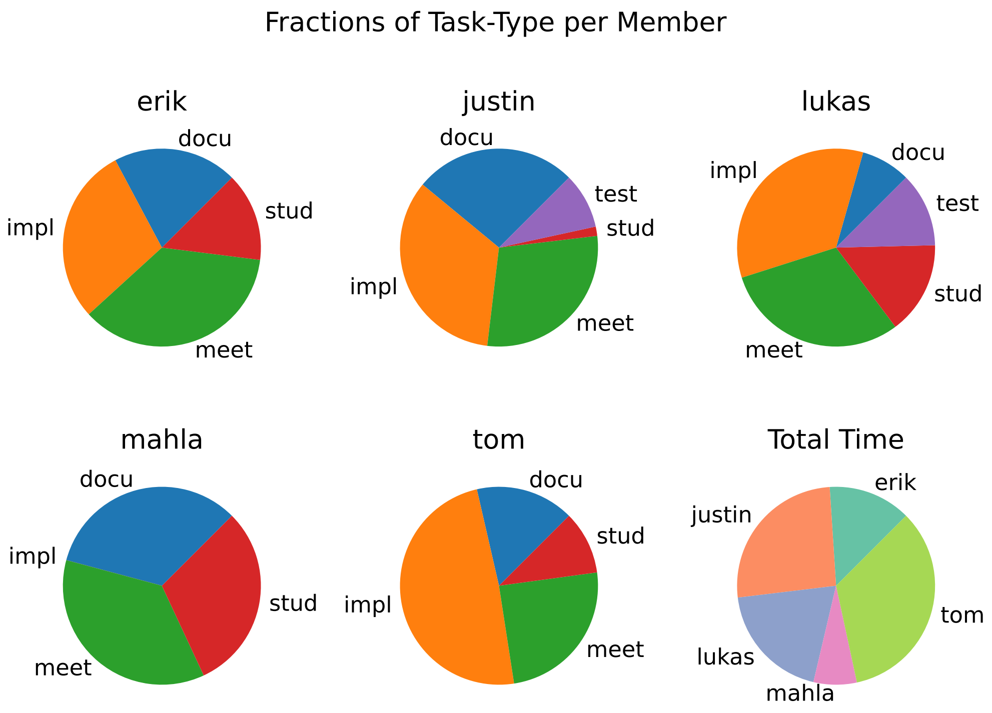


Figure 12: Distribution of time spent on project-related tasks per member.

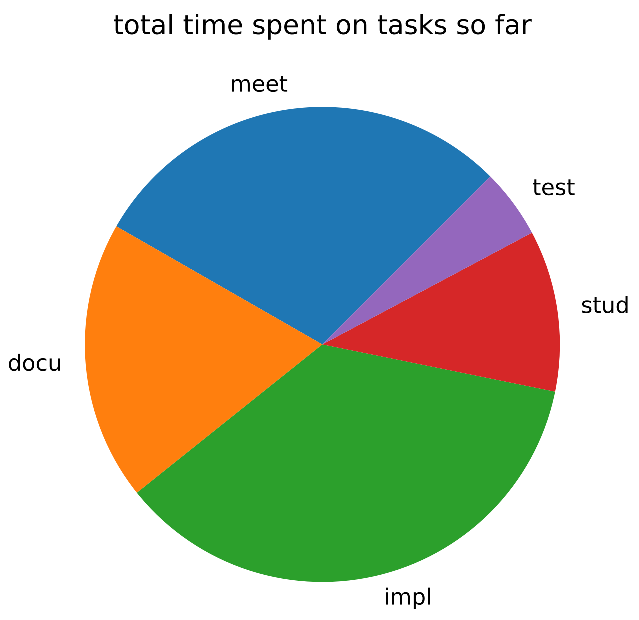


Figure 13: Distribution of time spent on project-related tasks in total.

References

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. “Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks”. In: *Knowledge-Based Systems* 8.6 (Dec. 1995), pp. 373–389. ISSN: 09507051. DOI: 10.1016/0950-7051(96)81920-4. URL: https://linkinghub.elsevier.com/retrieve/pii/0950705196819204 (visited on 11/11/2021).

[AK12] M. Gethsiyal Augasta and T. Kathirvalavakumar. “Rule extraction from neural networks: A comparative study”. In: *International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012)*. Salem, Tamilnadu, India: IEEE, Mar. 2012, pp. 404–408. ISBN: 978-1-4673-1039-0. DOI: 10.1109/ICPRIME.2012.6208380. URL: http://ieeexplore.ieee.org/document/6208380/ (visited on 11/14/2021).

[CS95] Mark W. Craven and Jude W. Shavlik. “Extracting thee-structured representations of thained networks”. In: 1995.

[Hai16] Tameru Hailesilassie. “Rule Extraction Algorithm for Deep Neural Networks: A Review”. In: *arXiv:1610.05267 [cs]* (Sept. 2016). arXiv: 1610.05267. URL: http://arxiv.org/abs/1610.05267 (visited on 11/14/2021).

[Joh07] Ulf Johansson. “Obtaining Accurate and Comprehensible Data Mining Models – An Evolutionary Approach”. In: (2007), p. 272. URL: https://www.diva-portal.org/smash/get/diva2:23601/FULLTEXT01.pdf.

[JKN10] Ulf Johansson, Rikard König, and Lars Niklasson. “Genetic Rule Extraction Optimizing Brier Score”. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation - GECCO ’10*. The 12th Annual Conference. Portland, Oregon, USA: ACM Press, 2010, p. 1007. ISBN: 978-1-4503-0072-8. DOI: 10.1145/1830483.1830668. URL: http://portal.acm.org/citation.cfm?doid=1830483.1830668 (visited on 11/08/2021).

[JM15] Enric Junque de Fortuny and David Martens. “Active Learning-Based Pedagogical Rule Extraction”. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.11 (Nov. 2015), pp. 2664–2677. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2015.2389037. URL: https://ieeexplore.ieee.org/document/7018925/ (visited on 11/08/2021).

[KZ15] Awudu Karim and Shangbo Zhou. “X-TREPAN : A Multi Class Regression and Adapted Extraction of Comprehensible Decision Tree in Artificial Neural Network”. In: *Computer Science & Information Technology (CS & IT)*. Academy & Industry Research Collaboration Center (AIRCC), Aug. 2015, pp. 37–54. ISBN: 978-1-921987-42-7. DOI: 10.5121/csit.2015.51405. URL: http://www.airccj.org/CSCP/vol5/csit54505.pdf (visited on 11/15/2021).

[MBV09] D. Martens, B.B. Baesens, and T. Van Gestel. “Decompositional Rule Extraction from Support Vector Machines by Active Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 21.2 (Feb. 2009), pp. 178–191. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2008.131. URL: https://ieeexplore.ieee.org/document/4564457/ (visited on 11/16/2021).

[OSF19] Seong Joon Oh, Bernt Schiele, and Mario Fritz. “Towards Reverse-Engineering Black-Box Neural Networks”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek et al. Vol. 11700. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 121–144. ISBN: 978-3-030-28953-9. DOI: 10.1007/978-3-030-28954-6_7. URL: http://link.springer.com/10.1007/978-3-030-28954-6_7 (visited on 11/09/2021).

[SW07] Emad W. Saad and Donald C. Wunsch. “Neural network explanation using inversion”. en. In: *Neural Networks* 20.1 (Jan. 2007), pp. 78–93. ISSN: 08936080. DOI: 10.1016/j.neunet.2006.07.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0893608006001730 (visited on 11/15/2021).

[SAG99] G.P.J. Schmitz, C. Aldrich, and F.S. Gouws. “ANN-DT: an algorithm for extraction of decision trees from artificial neural networks”. In: *IEEE Transactions on Neural Networks* 10.6 (Nov. 1999), pp. 1392–1401. ISSN: 10459227. DOI: 10.1109/72.809084. URL: http://ieeexplore.ieee.org/document/809084/ (visited on 11/17/2021).

[TP15] Mathieu Templier and Guy Paré. “A Framework for Guiding and Evaluating Literature Reviews”. In: *Communications of the Association for Information Systems* 37 (2015). ISSN: 15293181. DOI: 10.17705/1CAIS.03706. URL: https://aisel.aisnet.org/cais/vol37/iss1/6/ (visited on 11/11/2021).

[Thr95] Sebastian Thrun. “Extracting rules from artificial neural networks with distributed representations”. In: *Advances in neural information processing systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1995. URL: https://proceedings.neurips.cc/paper/1994/file/bea5955b308361a1b07bc55042e25e54-Paper.pdf.