University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's Thesis

Explainable Artificial Intelligence

Pilsen 2020 Bc. František Koleňák

Místo této strany bude zadání práce.

Místo této strany bude druhá strana zadání práce.

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, 10th August 2020

Bc. František Koleňák

Abstract

The thesis is focused on the Explainable Artificial Intelligence (XAI). Since Artificial intelligence (AI) is tightly coupled with (XAI), we will provide an overview of both. The overview consists of the history of both technologies and the introduction.

In the thesis is explained what leads the need for XAI. The thesis also contains some of the techniques used in XAI and libraries that implement such techniques.

The thesis includes experiments with libraries and compares the results to each other and other work.

Abstrakt

Diplomová práce je zaměřena na vysvětlitelnou umělou inteligenci. Vysvětlitelná umělá inteligence je pevně spřažena s umělou inteligencí, tedy v práci jsou popsány obě odvětví. Práce obsahuje popis historie a úvod do problematiky.

V práci je popsáno proč je nutné umět vysvětlit rozhodnutí umělé inteligence. Práce také obsahuje techniky používané pro vysvětlitelnou umělou inteligenci a knihovny, které používají popisované techniky.

Byly provedeny experimenty s knihovnamy a porovnány výsledky technik, jak mezi sebou, tak s ostatními pracemi.

Contents

1 Introduction

Although we have heard the term Explainable artificial intelligence (XAI) only recently, it is quite an old topic. The earliest work on XAI can be found several decades ago[98, 103]. At that time, the Artificial intelligence (AI) and Machine learning (ML) were still in the development, and the resources were very limited.

In the last decade, the AI and ML started to be used almost everywhere, due to cheap resources like computational power and data. Because of that, AI achieved (super) human performance (DeepBlue beating best chess player[6], IBM Watson winning in Jeopardy![42] or AlphaGo beating best player in Go[1]) and started to be used in many industries like healthcare, law, defense, finance or self-driving cars [29].

AI has played key role in many sci-fi movies where such a system surpassed its creators (e.g., Terminators Skynet[1] or A Space Odyssey[2]). This concern brought fear from AI into many people, including Stephen Hawking or Elon Musk [3], and started programs like DARPAs XAI [47], whose aim is to create AI that would be able to explain its decision (Figure 1.2).

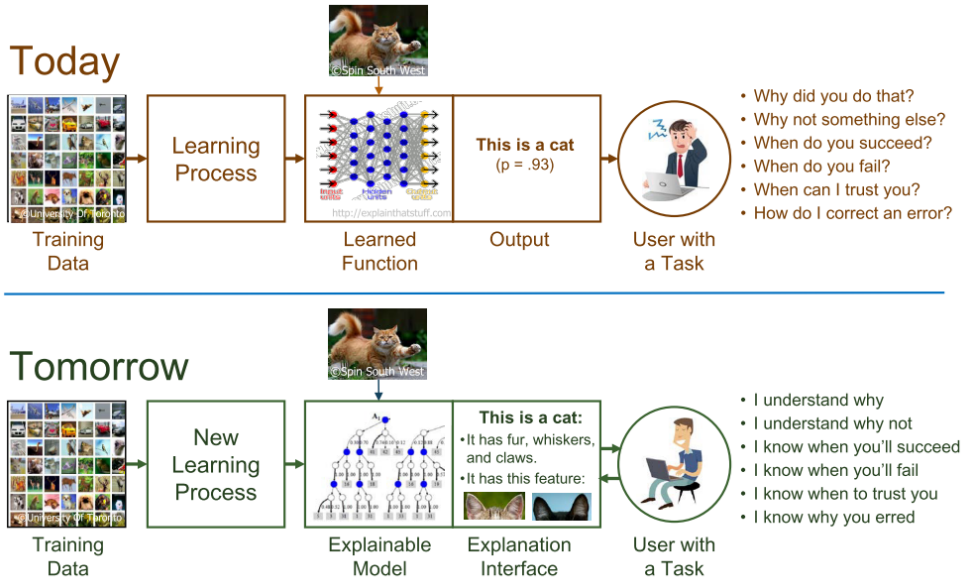


Figure 1.1: The future vision [47].

[1] https://en.wikipedia.org/wiki/Skynet_(Terminator)
[2] https://www.imdb.com/title/tt0062622/

XAIs aim is to create trust between machines and humans and create safer and more transparent AI that would explain its decisions to its users. The explanation is most important in industries where people's lives depend on the outcome of these systems. This topic also brings the AI ethic to life because we have to define the morality of the AIs decision. AI ethic is currently studied field where the researchers put people around the world and study their decisions on some moral problems[3] [27, 82].

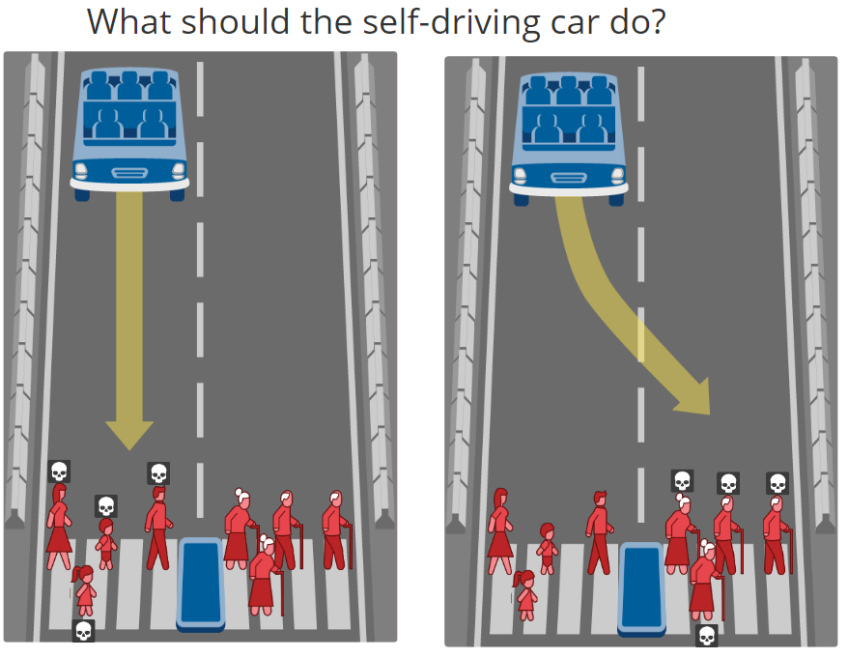


Figure 1.2: *Moral Machine* —Judge interface. This particular choice is between a family or old people [14].

In this thesis, we will introduce both AI and XAI, and point the importance of explainability and why it is crucial and beneficial to be able to understand the decisions of AI systems.

We will introduce some XAI algorithms that are used nowadays and search libraries or other services that could help us with the task of explaining AIs decisions.

Finally, we will demonstrate the use of XAI algorithms with the help of found libraries. We will discuss the output of the XAI and compare our results to other work.

[3]https://www.moralmachine.net

2 Artificial intelligence

The first definition of AI could be in 1950, where Turing [105] introduced a test, Turing test, to test a machine's intelligence. However, according to a book [34], the AI terminology was born at a workshop in 1956. Since then, the field was studied, but we have discovered only a tip of the possibilities. At first, there was a slow development, but in the 90's and early 21st century the computers started to have enough computing power and data for AI to be used in health and pharmaceutics, data mining, logistics, and other areas.

AI classification can be grouped by how close the AI can emulate human thinking. Thus we can differentiate how the system compares to humans in terms of versatility and performance. Based on these criteria, we classify AI systems on the similarity to the human mind and their ability to "think". This ability could be classified into four basic categories: reactive machines, limited memory machines, a theory of mind [101], and self aware [21] AI.

Reactive Reactive machine has no memory, only corresponds to different stimuli. It is the oldest type of AI system, and it is also the "simplest". These systems do not have any kind of memory (no ability to learn) and are very limited to what they can do. Often they are programmed to do only one specific thing, thus reacting to a limited set or combination of inputs. A popular example of reactive machine AI is DeepBlue[6], a machine that beast a master at chess.

Limited Memory Machines Limited Memory Machines have the same capabilities as reactive systems but uses memory to improve its responses. These systems learn from historical data, which are used to improve decisions. Nearly all of today's existing applications fall into this category (e.g., chatbot or self-driving car).

Theory of Mind A theory of mind-level AI will understand the entities it is interacting with by comprehending their needs, emotions, beliefs, and thought processes. It is still a concept or work in progress. To create AI of this level, researchers need to create a system that perceives humans as individuals whose minds can be defined by multiple factors [101].

Self aware AI Self aware AI This is the final stage of AI development, which currently exists only hypothetically. Self-aware AI has evolved to be so similar to the human brain that it has developed self awareness [21].

The ability to learn and improve from data was so significant that it formed another field within the AI called ML. ML shifted focus from achieving artificial intelligence to solving practical problems. It changed from the symbolic approaches and looked towards methods and models used in statistics and probability theory [67].

After a while, Le et al. [68] revisited the old concept (deep neural network), and they made the first unsupervised network in 2011. This network was trained to recognize higher-level concepts, such as cats, only from watching unlabeled images [68]. Deep neural network theory was first abandoned because researchers did not have enough computing power and data to prove it, hence why it was possible from the last decade. This research started the "boom" of Deep learning.

Deep learning is part of a broader family of machine learning (the relationship between AI, ML and Deep learning (DL) can be observed at Figure 2.1) based on Artificial neural networks (ANN) with representation learning. ANN are neural networks similar to biological neural networks, but their connections are static instead of dynamic.

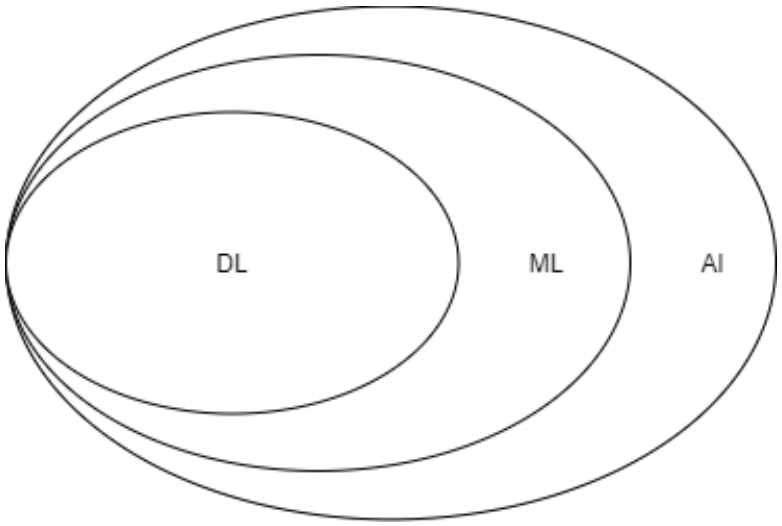


Figure 2.1: Relationship between AI, ML and DL

2.1 Machine learning

In 1959, defined Arthur Samuel the ML as the ability for computers to learn without being explicitly programmed. In practice, the programmer does not have to write every possible state. Instead, the task shifts to finding an algorithm that can extract patterns from the dataset and using the patterns

to build a predictive model that approximates a function that generalizes the data [96].

In this context, generalization means to perform accurately on new unseen data, after processing the training data set. The training data set is usually full of examples from some generally unknown probability distribution [31].

Because the training sets are not infinite, and the future is uncertain, learning theory does not yield guaranteed algorithm performance. Instead, the results are provided with bounds of certainty. The correctness can be improved with better training data, which is a topic by itself because the data can be under-fitted or over-fitted, so we need to provide balance to the training data set [22].

2.1.1 Types of learning

The learning process can have different approaches. Typical for most of the applications in ML is unsupervised and supervised learning. We will list only some of the learning techniques:

Supervised learning Supervised learning algorithm creates a mathematical model that contains both inputs and desired outputs. The supervised learning algorithm learns a function that can be used to predict the output associated with new inputs. The training data contains a set of features that include one or more inputs and their assigned label [28].

Unsupervised learning Unsupervised learning is the ability to find patterns in features without any human intervention. This means that the algorithm receives only inputs, and from those inputs, it is supposed to classify the data [28].

Semi-supervised learning This is the mix of both supervised and unsupervised learning algorithms. Researchers found that having a small amount of labeled data and unlabeled data can produce better results (learning accuracy) [62].

Reinforcement learning Reinforcement learning is an area of machine learning where software agents receive rewards for their actions. Agents gradually learn what actions produce the best reward [44].

Self-learning Self-learning is a learning with no external rewards and no external teacher advice [25].

Other learning techniques Other techniques include sparse dictionary learning [104], transfer learning [92], adversarial learning [81], feature learning [30] and so on.

2.2 Artificial neural network

Designed in 1943 by McCulloch and Pitts [73], ANNs were inspired by information processing in biological systems, but ANNs differs from live organisms. The biological brain of most living organisms is dynamic and analog, but ANN is static and symbolic. The structure in ANNs consists of interconnected nodes called neurons and set of edges that connect neurons [86].

Figure 2.2 shows one artificial neuron, that has three inputs to the neuron (x_i) and output y. Each input has weight assigned (w_i) and is connected to the neuron. Weight show significance of the input. Transfer (or threshold) function f computes the weighted sum ($\sum$)

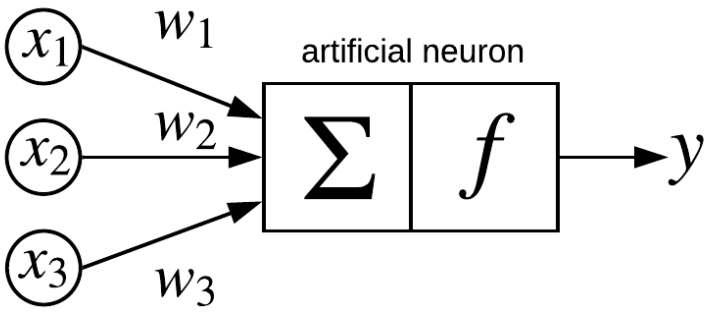


Figure 2.2: Artificial neuron.

The primary function of ANN is to receive a set of inputs and perform several procedures on input sets and use the resulted output to solve some problems [86, 97].

ANN tries to optimize the weights of the neurons so that the network has the best accuracy in interpreting the output based on a given input. Training ANNs can be done with the addition of backpropagation.

Backpropagation is a method that relays the information backward so that the algorithm can compute the gradient. This process (gradient descend) is then done iteratively to achieve the best accuracy [56].

2.2.1 Types of Artificial neural networks (ANN)

In ANN, there are a lot of architectures, each designed for a different type of tasks. Feedforward Neural Network (FNN) is one of the basic networks, and many architectures are built with principles from this network. We will provide just an overview of some of the architectures.

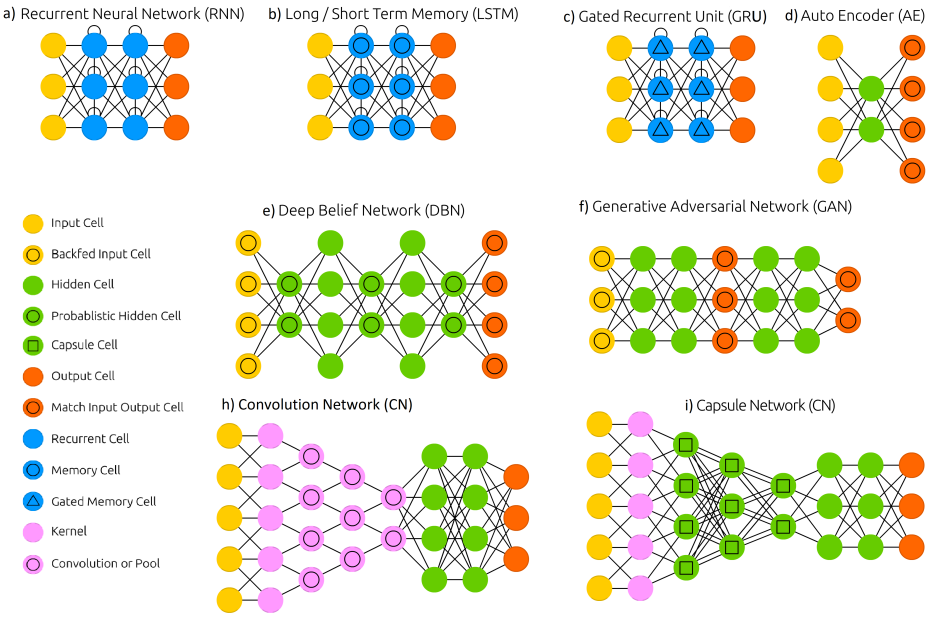


Figure 2.3: Types of neural network with their architecture[106].

Feedforward Neural Network (FNN)

FNN is a basic type of ANN that has a connection between neurons that do not form a circle (examples of FNN can be seen in Figure 2.5). Connections have a weight assigned to them, and they provide output to one neuron and input to another. Weight represents relative importance [86, 97].

Auto Encoder

Auto Encoder is a type of unsupervised (§2.1) neural network (Example in Figure 2.3d) that is typically used for reducing the dimensionality of data by removing redundant information. The network consists of reduction and reconstruction layers. The network includes an internal "bottleneck" layer of

a smaller dimension. Since the bottleneck layer has fewer neurons than the input layer, it has to represent or encode the information in the subsequent layers' inputs to reconstruct the input. Everything on the left side of the bottleneck layer is called the encoding part, and on the right is the decoding part. The result of the network bottleneck may identify only significant features of the data [64].

Deep Belief Network (DBN)

DBN (Example architecture Figure 2.3e) is a class of Deep neural networks (DNNs)(§2.3). The network has multiple layers, where the first two layers form an associative memory. Lower layers in the network receive input from layers above. The learning process is done layer by layer, where the output of one layer is training data for the layer below. DBN can be used for generating and recognizing images, videos, and motion-capture data [53].

Long / Short Term Memory Network (LSTM)

LSTM (Example architecture Figure 2.3b) introduces gates and explicitly defined memory neurons. Each neuron has a memory cell and three gates: input, output, and forget. These gates are used for regulating the data flow inside the network. The forget gates usage is to reset the information stored inside the memory. LSTM can be used for sequential data, like text or video [54].

Gated Recurrent Unit (GRU)

GRU (Example architecture Figure 2.3c) is slight variation of LSTM. Instead of three gates, they have two: update gate and reset gate. Reset gate has the same functionality as forget gate from LSTM. Update gate has two functions: first, it determines how much information to let in and how much to keep from the previous state. Since GRU does not have an output gate, they always send out their full state. GRU is, on smaller datasets, faster and less expensive to run than LSTM [32].

Generative Adversarial Network (GAN)

GAN (Example architecture in Figure 2.3f, red neurons in the middle are the output of one network and input for the other) is made out of two networks that work together. The first network is generating content for the discriminating (second) network. The discriminating network receives either training data or content from the first network, which is then evaluated and serves as input to the generating network. Both networks then compete with each

other, where the discriminating network is getting better at differentiating the learning and generates data. The generating network is getting better at generating data. This architecture can be difficult to train because if one of the models is much better than the other, the GAN will not converge [45].

Capsule Network (CN)

CN (Example architecture in Figure 2.3i, green neurons with square in the middle are capsule cells) is a special type of CNN(§2.2). CN adds structures called capsules (group of neurons) into the CNN which provides new alternative to pooling layer(§2.4.1). Capsules are outputting a vector instead of weight, allowing the transfer of more information, like position, color, or orientation of a feature [93].

Recurrent Neural Network (RNN)

RNN (Example architecture in Figure 2.3a) are FNN that depend on connections through time. Neurons are fed information from the previous layer as well as information from themselves from the previous pass. This means that the order of data the neuron is provided matters.

RNN suffers a problem where the information could get lost over time because the weight of the previous information could become 0 or really big, and such information has no meaning. Nevertheless, the occurrence of this phenomenon is dependent on the activation function (§2.2.3) used [41].

RNN is often used for advancing or completing information (autocomplete).

Convolutional neural network (CNN)

In deep learning, a Convolutional neural network (CNN) (Figure 2.3h) is a class of deep neural networks, primarily used for image analysis (can be also used for text classification [60]). More detailed description is in Section §2.4.

2.2.2 Loss functions

Loss functions for classification are computationally feasible functions representing the price paid for the inaccuracy of predictions in classification problems [91]. In other words, the results of loss functions are telling how accurate is the solution.

The commonly used functions are:

Mean Square Error/Quadratic Loss/L2 Loss Mean squared error loss functions are used to measure the accuracy of a model. Mean squared error is the average of the squared difference between predictions and observation. The smaller the number, the better the prediction is [85, 91].

Mean Absolute Error/L1 Loss Similar to mean squared errors, but are measured as a sum of absolute differences between predictions and observations [85, 91].

Cross-entropy Loss Cross-entropy loss function is often used for classification, where the network is classifying into binary class. The loss function has an output between 0-1. So we can classify value that is smaller than 0.5 as a "0"[85].

2.2.3 Activation function

An activation function is a neuron's property, a function of all the inputs from previous layers. The output of activation functions is the input for the next layer. Models' ability to process non-linear problems is dependant on the activation function [56].

Some of the types:

Softmax is usually at the output layer. Softmax will output probabilities for the output classes. Each component in the output will be between 0-1 (representing probability), and the components add up to 1. The component with the highest probability (number) is the predicted class [56].

Sigmoid is a squashing function. Squashing functions limit the output to a range between 0 and 1 [35].

ReLu is a linear function that will output the input directly if it is positive. Otherwise, it will output zero [79].

2.2.4 Optimizers

Gradient descent is one of the most popular algorithms for optimizing neural networks. Gradient descend has three variants [55, 61, 89]:

Batch gradient descent Batch gradient descent, computes the gradient of the cost function for the parameters for the entire training dataset. The calculation needs to be over the whole dataset to perform one update.

Stochastic gradient descent Stochastic gradient descent performs a parameter update for each training example.

Mini-batch gradient descent Mini-batch gradient descent takes the best of both worlds and performs an update for every mini-batch of n training examples.

Most known stochastic gradient descent algorithms are Adam [61] and Adadelta [55, 89].

2.2.5 Metrics

Metrics is a function that is used to judge the performance of a model. Since they are mostly derived from confusion matrix (§2.2.5), we will introduce it first, after that, we will look at the calculated metrics (§2.2.6).

Confusion Matrix

Confusion matrix [100] typically looks like in Table 2.1. Matrix can be used for any classification model. Typically, rows represent actual class and columns represent the predicted class.

		Predicted class	
		P	**N**
Actual class	**P**	TP	FN
	N	FP	TN

Table 2.1: Confusion Matrix

In binary classification, the classes are **P**ositive and **N**egative. This creates a 2x2 matrix containing these basic values:

TP True-Positive is the number of Positive cases classified correctly.

TN True-Negative is the number of Negative cases classified correctly.

FP False-Positive is the number of cases where the model predicted Positive class, but the actual class was Negative. Also known as Type 1 Error.

FN False-Negative is the number of cases where the model predicted Negative class, but the actual class was Positive. Also known as Type 2 Error.

2.2.6 Calculated Metrics

Using the basic values, different metrics can be computed:

Precision (2.1) Out of all predicted Positive cases, how many were correct.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

Recall (2.2) Out of all actual Positive cases, how many were correctly predicted, meaning how close were the measurements to each other.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Accuracy (2.3) is the most common performance metric for balanced datasets. It is a ratio of the correctly guessed predictions and the total number of predictions. It does not work well for an unbalanced dataset since it may give way to methods biased towards the majority class. Often used when every class is equally important [74].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.3}$$

2.3 Deep learning

DL is part of ML (§2.1) methods that utilize ANN (§2.2). Although the evolution of DL started between the 1940s and the 1960s, researchers did not have enough computational power and data to truly test the limits of the developed methods [23, 97].

The DL uses hierarchical level of the ANN to carry out the process of ML (§2.1). This structure then enables to the process of data with a nonlinear approach, as opposed to traditional programs that are built in a linear way (Figure 2.4) [23, 97].

The deep in the name of these learning methods comes from the use of multiple layers in the network. The layers consist of an input and output layer and a hidden layers between these two layers. There are also shallow neural networks that differentiate betweenDNN by the number of hidden layers. To be considered a shallow network, the network has to have up to two hidden layers (not a set standard, but most researchers agree). Example ANN for the shallow and deep neural network can be seen in Figure 2.5 [23].

DL can achieve higher accuracy because of the number of hidden layers that are added to the network, because of the high computational power,

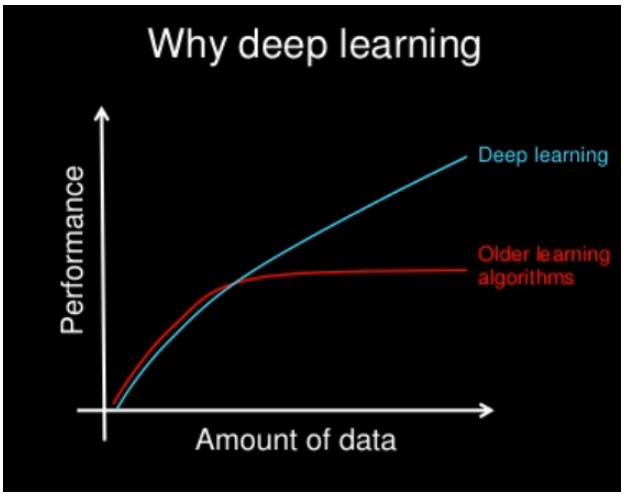


Figure 2.4: Differences in accuracy between DL and traditional programs[80].

and because of the large volume of data that is fed to it. The significant part is also the type of learning (§2.1.1) used [56], because it can enable us to use even more data (e.g., we do not have to label data). By having large datasets, the chance that the model has not seen some cases gets significantly lower, thus less chance of failing and higher accuracy.

In our experiments (§6) we will be using CNN, we will describe this type of DL network (DNN) in more detail (§2.4).

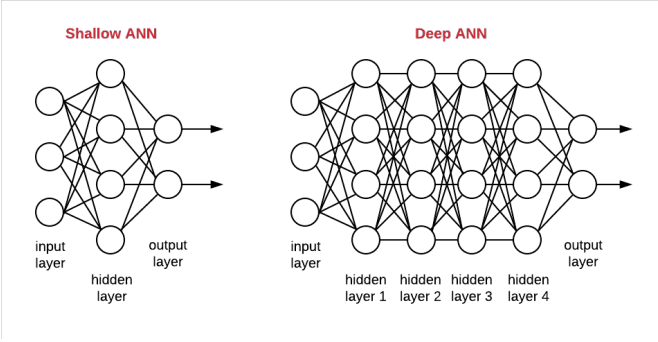


Figure 2.5: Example ANN for shallow and deep neural network

2.4 Convolutional neural network (CNN)

CNN takes raw input image vector and outputs single perceptive score function (weight). The last layer (§2.4.1) contains loss function(§2.2.2) which is affiliated to the classes. Since CNN is mostly used for images, we can encode image-specific features into the architecture [84].

Architecture of CNN consist of three basic types of layers (convolutional, pooling and fully-connected). When these layers are stacked, they form architecture of CNN. Figure 2.6 shows simplified architecture of CNN. Architecture in example includes all layer types (7 layers in total) and outputs probabilities for four classes.

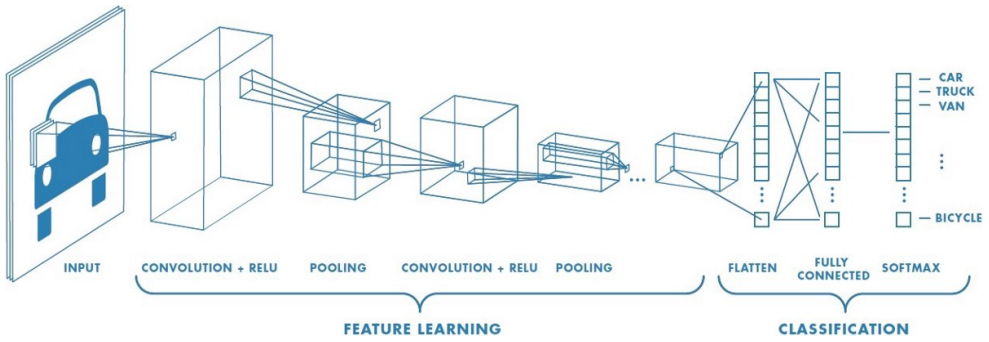


Figure 2.6: Example CNN with simplified architecture with convolutional, pooling and flatten layers. (Image courtesy of [4])

2.4.1 Layers

Although ANNs have many different layer types, we will list only some of the layers specific for CNN.

Convolutional layer

The objective of the convolutional layer is to extract low-level features, such as edges, color, or gradient orientation from the input image. With the addition of more convolutional layers, we can extract more precise features (high level), and the model can gain a better understanding of the image.

For the calculation, the rectified linear unit (ReLu) is used. ReLu aims to apply element-wise activation function (§2.2.3), such as sigmoid, to the output of the previous layer.

In practice, that means that we have a filter with three hyperparameters (depth, stride, and padding), and the filter goes all over the image, triggering kernels (when they see specific features). This triggering is commonly referred to as activation [84].

Pooling layer

Pooling layers are used to reduce the complexity of the model by reducing the number of parameters. This layer reduces the dimensionality of the

representation.

The pooling layer operates over each activation in the input and scales the dimensionality using max, average, or sum function. For example, the kernel of dimensionality 2x2 will reduce original activation down do 25% of the original size [84].

Fully-connected layer

The fully-connected layer contains neurons that are directly connected to the neurons in the two adjacent layers.

Flatten layer

Removes all of the dimensions of the input except for. E.g., from the shape (64,10) to shape (640).

2.5 Neural Network Frameworks

ANN (§2.2) frameworks are often used to implement applications that are used for ML. They implement models(§2.2.1), optimizers (§2.2.4) and metrics (§2.2.5) mentioned in §2.2 and many more features that are then used to create applications that use ML models (including DL models). Frameworks help developers speed up the process of creating models, by offering simplified API. Some of the frameworks:

Tensorflow [18] Tensorflow is open source library to help with development and training ML models.

Keras [11] Keras is open source interface used for running Tensorflow or Microsoft Cognitive Toolkit (CNTK). Keras simplifies the usage of libraries and includes helpful functions.

Pytorch [15] Pytorch is open source machine learning library developed by Facebook.

In our experiments (§6) we will be using CNN using framework Keras (with Tensorflow).

2.6 Trustworthiness of an AI

Even if we know the basic functionality of an AI and ANN (§2.2), we usually do not know what leads to the final decision, which is why these systems are

often referred to as "black boxes", that produce values that are interpreted to signify a certain meaning. While there may be no need for an explanation when such a system is used for predicting what movie I would like to watch next, there are certainly some fields where knowing why the system produced some value is crucial. If we start to question the AIs decision, especially if there is a system that influences people's lives, we raise questions like [48]:

- Why did you do that?
- Why not something else?
- When do you succeed?
- When do you fail?
- When can I trust you?
- How do I correct an error?

Suppose we are unable to answer these (or similar) questions. In that case, it can lead to a disaster for the company, government, or whoever uses such a system, since the system can be biased (or have different flaw) and we would not even know about it.

2.7 Explainability need

Instead of long theoretical discussions, let us demonstrate why some systems need to be explainable.

Self-driving car can potentially miss interpret some objects, and the car could crash into them. Who will be responsible for the crash, the passenger, software developer, or car manufacturer?

Cancer detection system's goal is to detect cancer. It could be disastrous if the system misinterprets the healthy tissue as cancer or misrecognize the cancer type as malignant whenever it benign. In this case, the patient could become depressed over nothing.

Mortgage system that is deciding if the person is eligible for a mortgage or not. Knowing why they did not get the loan can lead them to a path where they fix their problems.

Those problems are just a few examples where the human looks at the AI problem as a black box, and there is a need to make it white box and, if not entirely, then at least for AI explainability.

2.8 AI ethics

From the example of self-driving cars (§2.7), we can see that there could be potentially a problem of defining the car's behavior.

Researchers are now focusing on general human behavior, and they are gathering data on human decisions from all around the world. These research papers are guiding developers, how machines should make moral decisions. One of the experiments can be found on Moral Machine[1] [27, 82].

[1] https://www.moralmachine.net

3 Explainable artificial intelligence

Explainable artificial intelligence (XAI) is an old topic. The earliest work on XAI can be found several decades ago[98, 103]. ML hit high popularity in the last decade. With the availability of large datasets and more computational power, ML systems have achieved (super) human performance (DeepBlue beating best chess player[6], IBM Watson winning in Jeopardy![42] or AlphaGo beating best player in Go[1]) in a wide variety of tasks.

At first, there was no need to explain AIs decisions made by these systems, because, in the past, the AI algorithms were easily interpretable (§4.1). Nowadays, the algorithms used are complex, and even the most straightforward deep neural network cannot be easily understood. The need for an XAI started to be more apparent when these systems appeared in our everyday lives. Unfortunately, such systems were so complicated that no one could understand or predict the outcome of the given input. The need was most notable in industries where human lives depend on these systems' outcomes, like healthcare, law, defense, or finance [29].

XAIs algorithms are used to explain AIs (§2) predictions. At its core XAI is a subset of Artificial intelligence (AI), but that does not mean that we have to apply AI to AI, ultimately it is human-agent interaction problem. So XAI needs to be able to explain its decision efficiently. It depends on whom we explain such a decision (Examples on the difference is in Figure 3.1).

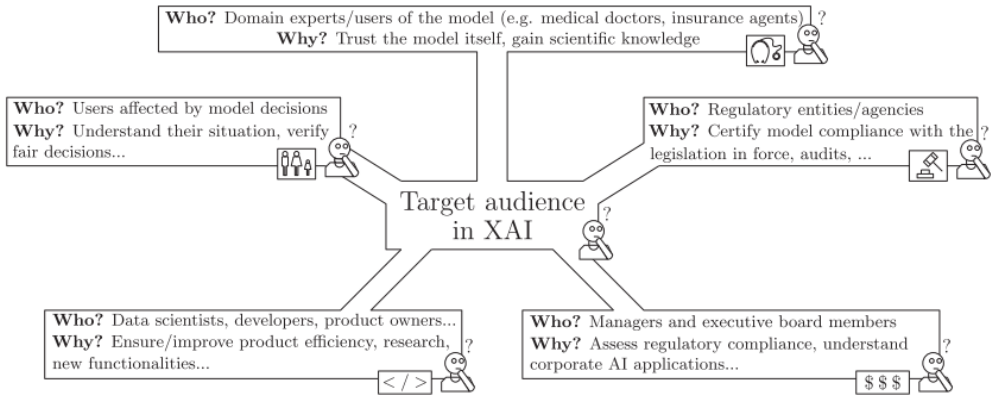


Figure 3.1: Explaining to different target audience [29].

3.1 Explainability / Interpretability Definition

Explainability and interpretability are often, in the context of machine learning, interchanged.

Interpretability is the degree to which humans can understand the cause of a decision. In other words, interpretability is about being able to discern the mechanics without necessarily knowing why.

Explainability is the extent to which the internal mechanics of a machine or deep learning system can be explained in human terms. Explainability is being able to quite literally explain what is happening.

Because of how close these terms are, in this thesis, the terms explainability and interpretability will be used interchangeably[33, 76].

For a system to be better, there should be a consideration for interpretability because it helps ensure unbiased decision-making, i.e., to detect, and consequently, correct from bias in the training dataset. Interpretability can act as insurance that only important variables infer the output, i.e., guaranteeing that an underlying truthful causality exists in the model reasoning[49].

3.2 Explainability Approaches

Explainability can be focused on different aspects of ML. We can focus on data explanation, where we try to understand the features in the dataset. We can focus on the model that uses the dataset and explain its decisions. Such a model can be directly interpretable(§4.1), or we can explain it post hoc.

Directly interpretable means the model itself is constructed so that we can explain its decisions(also called white boxes).

Post hoc means that we are explaining the model after it is constructed. After constructing the model, we probe into the model with a companion model to create interpretations (Figure 3.2)[26, 95].

Post hoc approaches can be differentiated by the scope they are taking a look at. Some approaches aim to explain the model as a whole. This approach is called a global interpretation. The opposite of global interpretation is local interpretation[95].

We can also differentiate if the explanation is static or interactive. The **static explanation** is unchangeable, so such an explanation can not be

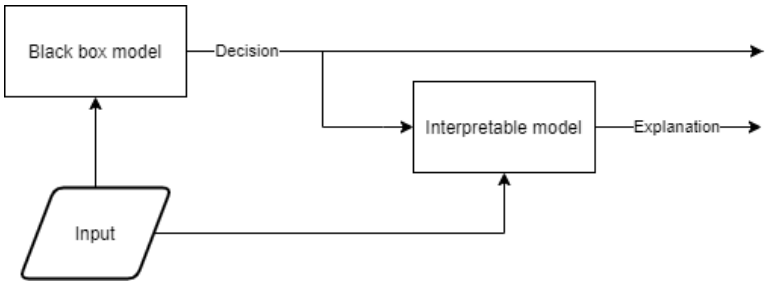


Figure 3.2: Example of post hoc architecture.

changed. **Interactive**, on the other hand, can be changed, allowing users to ask for a different explanation or provide a more in-depth explanation until the user is satisfied with the explanation[26].

Algorithms for explanations can be aimed at one specific model, or they can be model-agnostic. **Model-agnostic** means that the algorithm does not care which model it is explaining[29].

In this thesis, we will focus on model interpretability with static interaction. An overview of how we can explain can be seen in Figure 3.3. Explanation based on samples means that we explain based on one data point. Explanation based on feature is taking the whole feature into account.

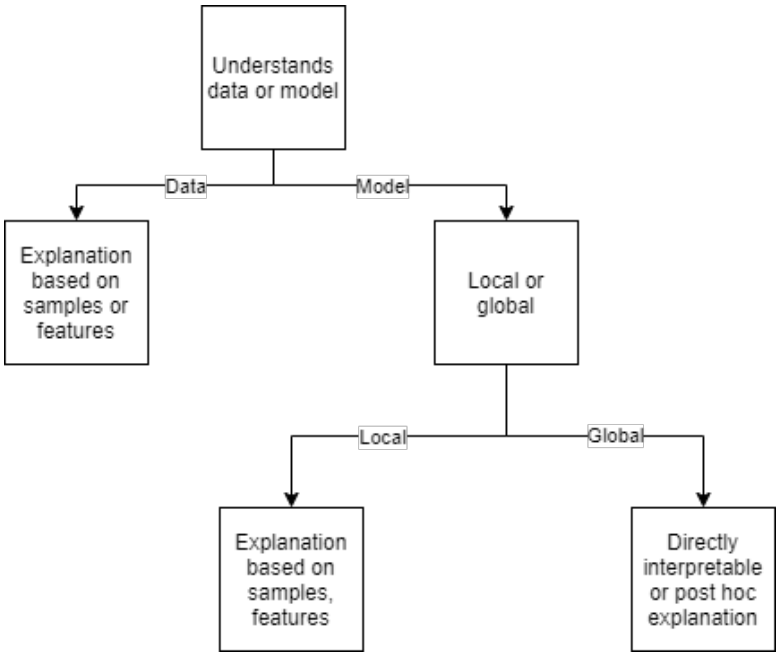


Figure 3.3: Explainability approaches

If we look at the scope in more detail we can differentiate the methods in following groups:

- Algorithm transparency (§3.2.1)
- Global model interpretability (§3.2.2)
- Global model interpretability on a modular level (§3.2.3)
- Local Interpretability for a Single Prediction (§3.2.4)
- Local Interpretability for a Group of Predictions (§3.2.5)

3.2.1 Algorithm transparency

Algorithm transparency is about how the algorithm creates the model, how it trains a model from the data, and what kind of relationships it can learn, which means that the inputs and the algorithm's usage must be known. This understanding is about knowing the algorithm but not about the learned model created in the end and not about how the individual predictions are made[39].

In this thesis, we will focus on the interpretability of the trained models and decisions made by these algorithms.

3.2.2 Global model interpretability

Global model interpretability tries to explain the behavior of the whole model. We can describe the model as interpretable if we can comprehend the entire model at once. Interpretation with this method starts by identifying prototypical cases for the output quantity (3.1) and allowing in principle to verify that the function has a high value only for the valid cases.

$$x^* = argmax_x f(x) \tag{3.1}$$

This approach cannot explain which features are essential and what interactions are happening between them. Achieving global model interpretability is very difficult because humans cannot easily hold in memory large amounts of data, so the model with many features will likely still be too complicated to explain[77, 95].

3.2.3 Global model interpretability on a modular level

Similar to global model interpretability, but divides the whole model into smaller parts. By doing that, we can understand parts of the model and, in

doing so, understand the model. If we take the Naive Bayes model with many features, we cannot efficiently work with so many variables. If we split this model into a single weight, we can easily understand and work with such information. While global model interpretation might be too complicated for us, this approach will make models understandable, at least on a modular basis. That said, not all models can be split into smaller parts. For example, linear models have weights interconnected to each other, and we cannot isolate one single weight[77].

3.2.4 Local Interpretability for a Single Prediction

Local interpretability for a single prediction is taking a look at the model's output for one specific instance(e.g., single data point or feature). In the explanation, it should highlight features that are relevant to it. By focusing on a single instance, the otherwise complex model might become easier to understand. This method might reveal more straightforward linear or monotonic dependence on some features, instead of having complex reliance on them[77].

3.2.5 Local Interpretability for a Group Prediction

Local Interpretability for a Single Prediction is similar to local interpretability for a group of predictions. However, instead of one investigation of why the model made a specific decision for one instance, it examines it for a group of instances. Model predictions for multiple instances can be explained by global model interpretation (on modular level) or with explanations of individual instances. By taking a group of instances and applying global methods, we can treat the group as a complete dataset and explain this subset. Another technique is to explain each instance and then list or aggregate results for the group [77].

4 Techniques

In this chapter, we will take a look at the most notable methods.

The simpler the model the more interpretable it is, but also less accurate (Figure 4.1). Most interpretable methods (models) will be located in section with interpretable models (§4.1), in sections Local model interpretability(§4.2) and Global model interpretability (§4.3) will be algorithms that are suitable to explain DNNs.

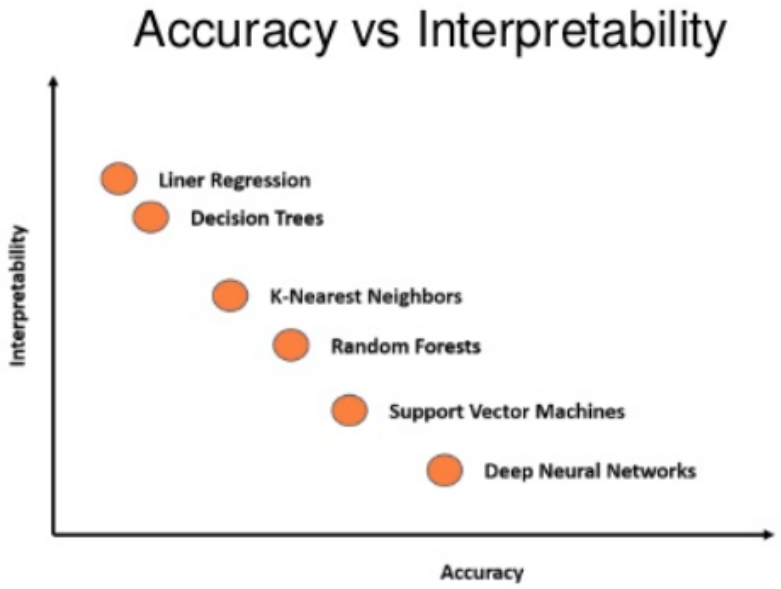


Figure 4.1: Accuracy vs interpretability of a model [94].

4.1 Interpretable models

In this section, we will see that model itself could be self-explanatory. These models are made by a subset of algorithms that create interpretable models. These models are usually widely known, so we will not go into details. Models that are interpretable by design are called Ante-hoc Explainability (AHE). Interpretable models can be examined for: safety/reliability, fairness/lack of bias, causality, or robustness [40].

4.1.1 Linear Regression

The term regression and its evolution primarily describe statistical relations between variables. The simple regression is the method that explores the relationship between one dependent variable and one independent variable. The simple linear regression model can be stated as: $y = \beta_0 + \beta_1 x + \epsilon$, where y is the dependent variable, β_0 is the y intercept, β_1 is the slope of the simple linear regression line (represents the learned feature weights or coefficients), x is the independent (explanatory) variable and ϵ is the random error. This simple linear regression model can then be enhanced with more features. This process is called multiple linear regression, and the final notation will be: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \epsilon$. From this notation, we can predict the outcome of an instance by summarizing its n features. To find optimal weights $(\beta_1 \dots \beta_n)$ we can use various methods. Usually, the least squares method is used to find the weights to minimize the squared differences between the actual and the estimated outcomes [43].

To make the "correct" version of the model, the data's relationships have to meet certain assumptions, which are linearity, normality, homoscedasticity, independence, fixed features, and absence of
multicollinearity [43].

Linearity This means that the response variable's mean value is a linear combination of the parameters (regression coefficients) and the predictor variables.

Normality It is assumed that the target outcome given the features follows a normal distribution. If this assumption is violated, the estimated confidence intervals of the feature weights are invalid [77].

Homoscedasticity This means that different values of the response variable have the same variance in their errors, regardless of the values of the predictor variables [77].

Independence This assumes that the errors of the response variables are uncorrelated with each other.

Fixed features The input values are considered as "fixed". The feature is considered fixed if they are treated as given values instead of statistical variables.

Absence of multicollinearity Having strongly correlated features is problematic because it becomes hard to estimate the weights. In this situ-

ation, the feature effects are additive, and it becomes unclear to which feature the attribute belongs to.

Interpretation

A fitted linear regression model can be used to identify the relationship between a single explanatory variable (x_j) and a dependent variable (y) while other explanatory variables are fixed. The interpretation of weight in the linear regression model depends on the type of the feature (e.g., numerical, binary or categorical) [43].

In Figure 4.2 we can see visualization of some example data. The black line is fitted linear regression line.

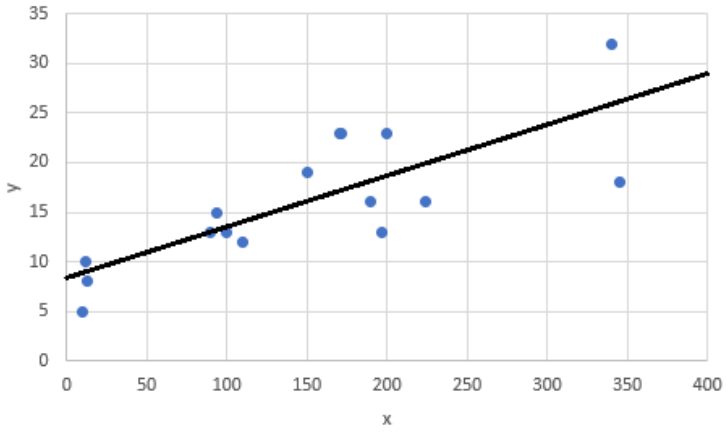


Figure 4.2: Example of linear regression.

4.1.2 Logistic Regression

Logistic regression is used to categorize the dependent (outcome) variable (predict if the email is spam or not).

The most common example of modeling is linear regression model (§4.1.1), where the dependent variable is assumed to be continuous. The logistic regression differentiates from the linear model (Figure 4.3) because the outcome variable is binary (it is also possible to have multiple classes). Logistic regression has three types [46]:

Binary logistic regression This type classifies into two possible outcomes.

Multinomial logistic regression Classifies for three or more categories without ordering.

Ordinal logistic regression Classifies for three or more categories with ordering.

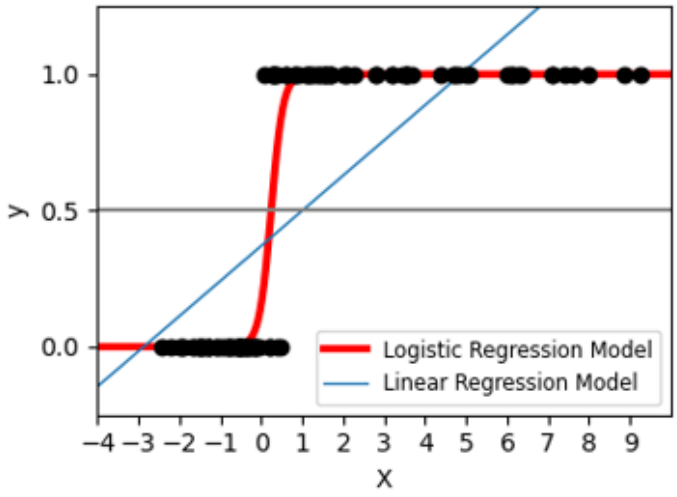


Figure 4.3: Example of linear regression and logistic regression. Input data are the black dots. Note that logistic regression is in binary range (0-1)[17].

Interpretation

The interpretation of any fitted model requires to be able to draw practical inferences from the estimated coefficients in the model. For most models, we need to know the estimated coefficients for input variables. The estimated coefficients represent the slope (i.e., rate of change) of a function. Thus for interpretation, we need to determine the relationship between input and output variables and define the unit of change for input variable[46].

4.1.3 Decision Tree

Decision tree (Tree example in Figure 4.4) is a tree-like structure where each node represents a test on an attribute, each connection represents the outcome of the test, and each leaf node represents the class label.[58, 87]

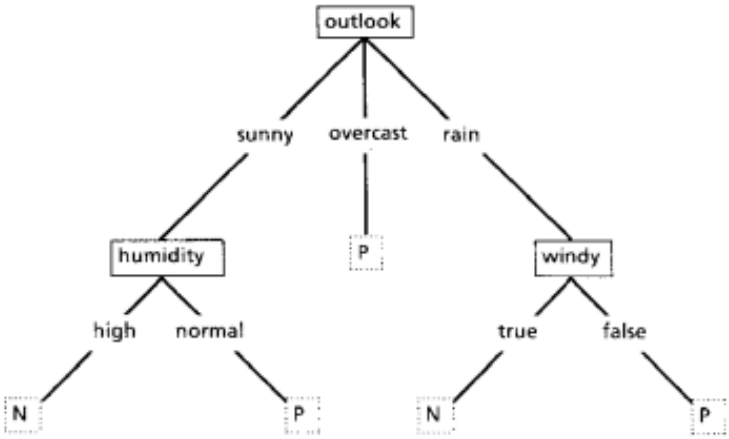


Figure 4.4: Example simple decision tree[87].

Decision Rules

The decision tree can be approximated if we linearize the tree into decision rules. Assembly of the rules follows this procedure: The content of the leaf node is the outcome, and the conditions that we followed along the path form a set of conditions. The set of conditions then form a conjunction in the if statement (*if* condition *and* condition2 *then* outcome) [88].

Interpretation

Interpretation of small decision trees is straightforward because the only thing we have to do is plot the tree. We might lose the interpretability for big trees, but the model is still transparent because we can follow the path.

4.2 Local model interpretability

Local model interpretability includes techniques that uses principles discussed in §3.2.4 and §3.2.5.

4.2.1 Local Interpretable Model-Agnostic Explanations

Local Interpretable Model—Agnostic Explanationss (LIMEs)[90] overall goal is to identify an interpretable model (can include any model from §4.1) over the interpretable representation that is locally faithful to the classifier. LIME is model-agnostic, so it treats every model as a black box, thus should be

able to explain any model. LIME can be used for text data, tabular data or images.

To help us understand how the original model behaves, LIME needs to use a representation of data, that we can easily recognize. For example, explaining the image classification, one way of showing the explanation can be a vector with pixels, or super-pixels (cluster of pixels, for example, in Figure 4.5), that have their values as present or missing.

Figure 4.5: Showcase of image which is divided into super-pixels [110].

Super-pixel is a cluster of pixels with similar color and brightness. A similar approach can be used for text classification, where the output can be a vector with a pair of words and values, where values indicate if the word is present or missing. Both approaches can then be used to visualize and pinpoint, which features are mostly used as a deciding factor for the classifier. Description of the LIME algorithm steps can be seen in Figure 4.6 [90].

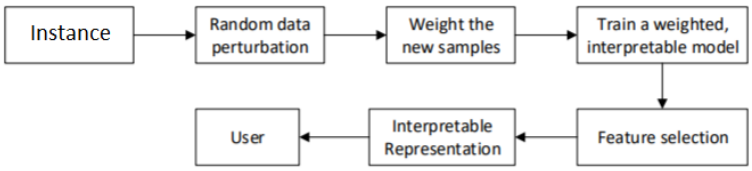


Figure 4.6: LIME framework diagram.

Example

Simplified example:

1. Select instance (image)
2. Divide the image into super-pixels (Figure 4.5).

3. Create new dataset with perturbed images (Essentially create similar images)
4. Tests the predictions by turning the super-pixels on or off (Figure 4.7a)
5. Train a weighted, interpretable model (Figure 4.7b)
6. Select the best feature (perturbed image) by interpreting the trained model
7. Show result (Figure 4.7c)

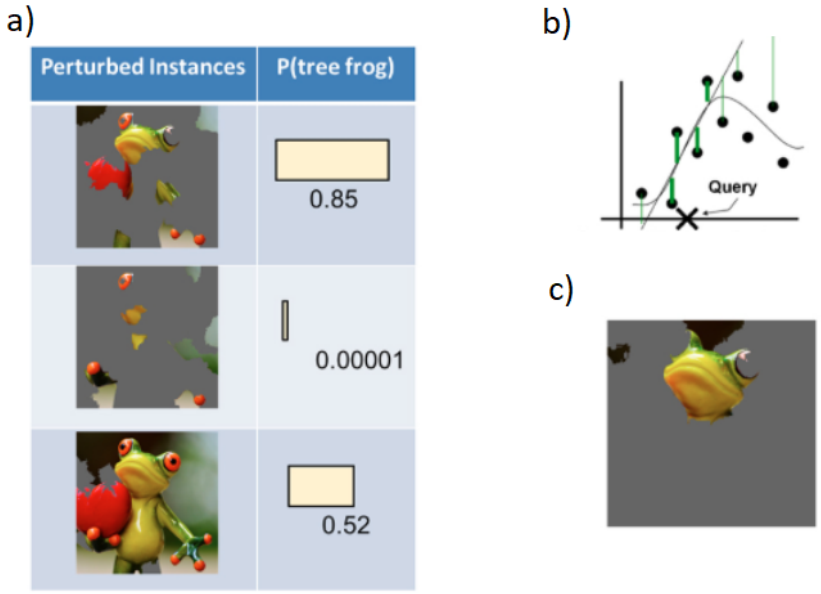


Figure 4.7: *a)* Random data perturbation step with weight assigned, *b)* Weighting and sampling instances, *c)* Select and show the super-pixels with best prediction [110].

4.2.2 Shapley values

Shapley values is a solution concept in the cooperative game theory of fairly distributing both gains and costs to several actors working in a coalition. The Shapley value applies in situations when each actor's contributions are unequal, but each player works in cooperation with each other to obtain the gain or payoff. Although the original work is designed for use in games, we can use Shapley value for a prediction task by defining the "game" as the prediction task for a single instance of the dataset. The "gain" as the actual

prediction for this instance minus the average prediction for all instances and the "players" as the feature values of the instance that collaborate to receive gain. To calculate Shapley value, we need the average marginal contribution of a feature value across all possible coalitions. Since computing, all possible coalitions can be very time-consuming. We can simplify this task to compute only a few samples of the possible coalitions. [51]

Interpretation of the Shapley value for feature value n is: The value of the n-th feature contributed to the prediction of this particular instance compared to the average prediction for the dataset. The Shapley value is the average contribution of a feature value to the prediction in different coalitions [77].

4.2.3 SHapley Additive exPlanations (SHAP)

SHapley Additive exPlanations (SHAP) first introduced by Lundberg and Lee [70] and is based on Shapley values (section 4.2.2).

The goal of SHAP is to explain the prediction of an instance x by computing the contribution of each feature to the prediction.

The SHAP method computes the Shapley values. In this case, the feature values act as "players" in a coalition. Shapley values then tell us the fair distribution among the features. Similar to LIME, the explanation is represented as an additive feature attribution method, as a linear model [70].

Additive feature attribution method is described in Equation 4.1. the $f(x)$ is the original model, $g(x)$ is the explanation model, $x^{'}$ is simplified input (has several omitted features), such that $x = h_x(x^{'})$. $\phi_0 = f(h_x(0))$ represents the model output with all simplified inputs missing. ϕ_i is the feature attribution for a feature i.

To compute Shapley values, we simulate that some features are present and some are absent. The representation as a linear model of coalitions for the computation is actually the trick of computing the ϕ_i's. For the feature of interest, we can omit every feature that is absent, so we can then simplify the formula to Equation 4.2.

$$f(x) = g(x^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i x_i^{'} \tag{4.1}$$

$$f(x) = g(x^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i \tag{4.2}$$

The properties of ϕ_i's have to satisfy some properties to be considered Shapley values:

Local accuracy When approximating the model for a specific instance, local accuracy requires the explanation model to at least match the output of the model for the simplified input x'.

Missingness Missingness means that the feature that is missing gets attribution of zero, so such a feature should not have any impact.

Consistency Consistency states that if a model changes so that some simplified input's contribution increases or stays the same, then the Shapley value should also increase or stay the same.

Deep SHAP

Deep SHAP combines Shapley values(§4.2.2) and DeepLIFT[99]. DeepLIFT linearizes non-linear components of a neural network. Deep SHAP adapts DeepLIFT to become a compositional approximation of SHAP values. Deep SHAP combines SHAP values computed for smaller components of the network into SHAP values for the whole network (§3.2.5) [70].

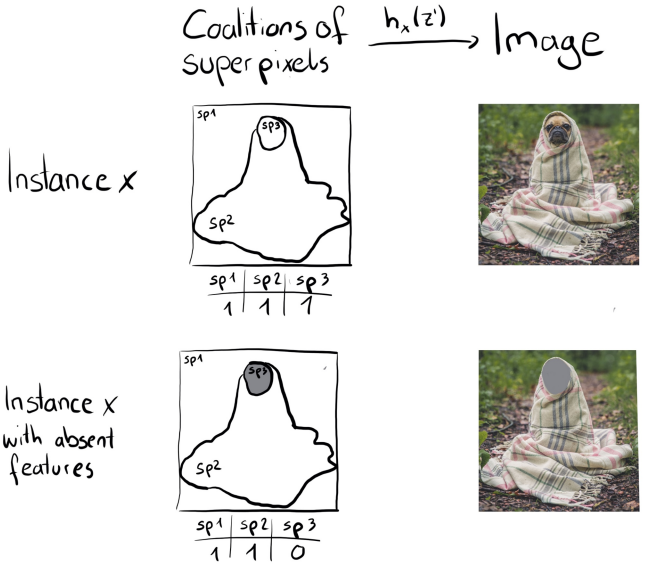


Figure 4.8: Example of Kernel SHAP with original instance X and instance with absent super-pixels [77].

Kernel SHAP

Kernel SHAP (Example in Figure 4.8) combines two different ideas:
LIME (§4.2.1) and Shapley values (§4.2.2). Kernel SHAP uses linear regression model (linear LIME model), and an appropriate weighting kernel to locally approximate function *f*. The regression coefficients of the LIME model estimates the SHAP values. Kernel SHAP is model agnostic method, but offers also model specific variant [70].

Tree SHAP

Introduced in the research paper [71]. This version of SHAP aims to explain tree-based ML models such as random forests, decision trees(§4.1.3), and gradient boosted trees. The proposed class *TreeExplainer* is also part of the library SHAP(§5.1.3).

TreeExplainer enables new local explanation method, which builds on top of Shapley values(§4.2.2). TreeExplainer enables the exact computation of optimal local explanations for tree-based models, extends local explanations to capture feature interactions directly, and provides a new set of tools for understanding global model structure based on many local explanations[71].

4.2.4 Teaching explanations for decisions (TED)

Teaching explanations for decisions (TED) provides meaningful explanations that match the mental model of the consumer. The meaningful explanation is information that provides an understandable explanation to the target audience, is actionable and flexible enough to support various technical approaches[52, 66]. This algorithm can be used for tabular, textual, and image data.

TEDs model produces both decision and an explanation, instead of exposing models parameters of how the model produces a decision. The explanation provided can be adapted for the targeted audience[52].

In Figure 4.9, we can see the architecture of the proposed algorithm. Notice the different inputs, that include the explanation (in any form), and output of the whole architecture, which includes the classification and explanation (Merged in training time).

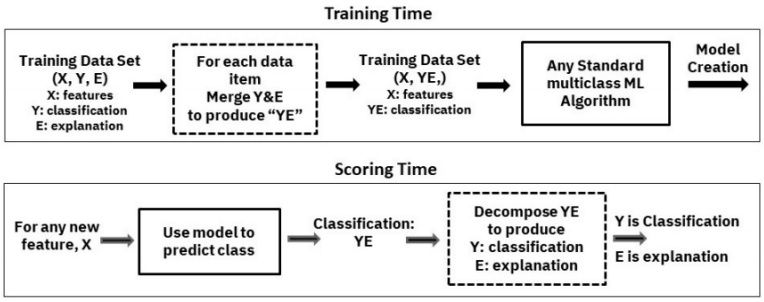


Figure 4.9: Overview of TED algorithm[52].

4.3 Global model interpretability

Global model interpretability includes techniques that uses principles discussed in §3.2.2 and §3.2.3.

4.3.1 Boolean Decision Rules via Column Generation

Boolean Decision Rules via Column Generation creates a globally interpretable model for binary classification and is used for tabular data.

The model uses Boolean rules in either disjunctive (DNF) or conjunctive (CNF) normal form. DNF classification rules are also referred to as decision rule sets, where each conjunction is considered a rule. At least one rule has to be satisfied to be considered as a positive prediction[36].

The method uses integer programming formulation for the Boolean rule (DNF or CNF) learning. It uses the large-scale optimization technique of column generation (CG) to search over all possible clauses, so that only useful clauses are generated or selected. CG starts with a small number of clauses, and iteratively adds missing clauses (pricing problem) until there are no improving clauses[36].

This method is best for small datasets, where it has the best interpretability and accuracy. Larger datasets might loose those aspects and add high computational time[36].

4.3.2 ProfWeight

ProfWeight is post hoc explanation algorithm (§3.2), which enhances the performance of another simple white-box model (lasso, decision trees, etc.). The idea is to have high performing DNN model and create an interpretable "copy" of that model. The implementation of this algorithm can be found in AIX360 (§5.1.1). This algorithm can be used for tabular, textual, and image

data.

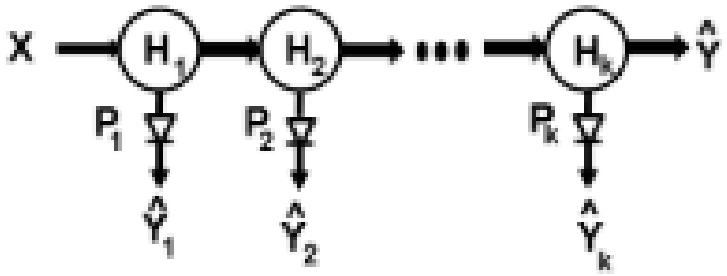


Figure 4.10: Network with added probes (in the picture marked as diode with label P_k). Probes have output $\hat{Y}_k$ and are attached to layers H_k [38].

The algorithm adds probes into the intermediate layers of original pre-trained DNN (Figure 4.10). Probes' goal is to obtain original models' predictions at the layer the probe is located. The probe is essentially a linear model with bias, followed by a softmax activation function [38].

The confidence scores (Figure 4.11) is a plot of the correct label of input at each of the probe outputs. The confidence score forms a curve that is called a confidence profile for that input [38].

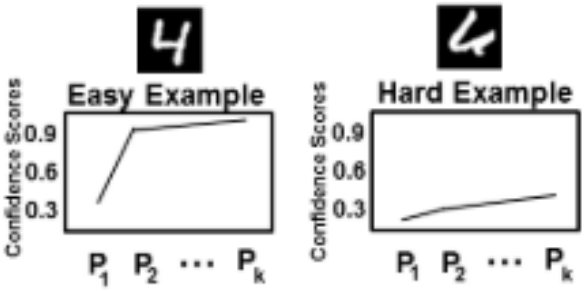


Figure 4.11: Confidence scores for two examples. Left example has high confidence score and is considered as "easy" and example on the right is considered "hard" [38].

When the probes are planted, the algorithm takes the confidence scores and feeds it to the neural network, which outputs an optimal weighting. The primary function of this process is to identify examples that the model will most likely fail on. The simple model ignores the "hard" examples. To identify the potential "hard" example, the algorithm computes the area under the confidence score [38].

To summarize, the algorithm is performing following steps [38]:

- Attach and train probes on intermediate representations of a high performing neural network.
- Train a simple model on the original dataset.
- Learn weights for examples in the dataset as a function of the simple model and the probes.
- Retrain the simple model on the final weighted dataset.

5 Libraries

This section lists libraries that we have found with our search. First we will describe some of the libraries, and then, we will compare the libraries.

5.1 Model interpretability

This section will have libraries that focus more on the model interpretation rather than model debugging or visualization.

5.1.1 AI Explainability 360 (AIX360)

AI Explainability 360 (AIX360)[1] is a toolkit developed in Python. This toolkit was designed to include multiple algorithms that help with explainability. The toolkit offers API, which unifies the interface for each interpretable algorithm, thus simplifying the toolkit usage. It is also prepared for custom implementation so that the user can add his algorithms. [26]

AIX360 contains multiple algorithms with different aims. Some focus on explaining data or model. Those algorithms that focus on model are then differentiated by how they explain and what is the scope they look at. As of now AIX360 contains these algorithms:

- ProtoDash [50]
- Disentangled Inferred Prior VAE [65]
- Contrastive Explanations Method [37]
- Contrastive Explanations Method
  with Monotonic Attribute Functions[72]
- Local Interpretable Model—Agnostic Explanations (LIME) - described in §4.2.1
- SHapley Additive exPlanations (SHAP) - described in §4.2.3
- Teaching explanations for decisions (TED) - described in §4.2.4
- Boolean Decision Rules via Column Generation (Light Edition) - described in §4.3.1

[1] Located at `http://aix360.mybluemix.net`

- Generalized Linear Rule Models[108]
- ProfWeight - described in §4.3.2

5.1.2 Local Interpretable Model—Agnostic Explanations (LIME)

The used algorithm is described in §4.2.1. The code was introduced together with the research paper [90]. LIME package is written in Python and is available on GitHub[2] or can be installed via Pip[3]. The package supports explaining text, tabular data, or images.

LIME package can use any classifier that implements a function that takes in raw text or NumPy array and outputs a probability for each class. The package also contains a scikit-learn classifier.

Authors state that they want to add more features to this package, but as of now, it supports only LIME algorithm with the above limitations and features.

5.1.3 SHapley Additive exPlanations (SHAP)

Used algorithm is described in §4.2.3. The code was also released with the paper by Lundberg and Lee [70] and is available at GitHub[4] or via Pip[5].

The main package is written in Python and offers various explainers. These explainers usually have some focus which makes them better for some tasks:

TreeExplainer TreeExplainer is an implementation of Tree SHAP (§4.2.3) and is used to compute SHAP values for trees.

DeepExplainer DeepExplainer is an implementation of Deep SHAP (§4.2.3) and is used to compute SHAP values for deep learning models.

GradientExplainer GradientExplainer is used for deep learning models for computing approximate SHAP values.

LinearExplainer LinearExplainer is for linear models.

[2] `https://github.com/marcotcr/lime`
[3] `https://pypi.org/project/pip/`
[4] `https://github.com/slundberg/shap`
[5] `https://pypi.org/project/shap/`

KernelExplainer KernelExplainer is an implementation of Kernel SHAP (§4.2.3) which is model agnostic method to compute SHAP values for any model.

5.1.4 Skater

Skater is Python package available at GitHub[6] or via Pip[7]. The project started as a research idea and incorporated algorithms that interpret models both locally or globally. Skater also includes naturally interpretable models. At the repository, they offer many examples. [16]

5.1.5 XAI

XAI is a machine learning library that was designed by The Institute for Ethical AI and ML. Package is available at GitHub[8] or via Pip[9]. The library has been designed to analyze and evaluate data and models and was developed to emphasize AI explainability. XAI implements a glass box model that has many visualization functions. XAI, at this date, is still in its alpha version (0.0.5). [7]

5.1.6 InterpretML

InterpretML is a package available at GitHub[10] or via Pip[11].

This package offers an interpretable model designed and developed for this package by Nori et al. [83] in their research paper. The research paper drafts this package and the design of the Explainable Boosting Machine (EBM). EBM is an interpretable model that can be as accurate as black-box models. Other than that, InterpretML also offers other interpretable models, black-box models, and tools for explaining black-box models. The package offers a unified interface, so interchanging models or explainers is much more comfortable. [83]

5.1.7 Others

This section lists other libraries that focus on interpreting models. We will not go in detail for each library:

[6] `https://github.com/oracle/Skater`
[7] `https://pypi.org/project/skater/`
[8] `https://github.com/EthicalML/xai`
[9] `https://pypi.org/project/xai/`
[10] `https://github.com/interpretml/interpret`
[11] `https://pypi.org/project/interpret/`

Teller Model-agnostic tool for ML explainability[78]

TreeInterpreter Package for interpreting scikit-learn's decision tree and random forest predictions. [20]

Alibi Alibi is ML library that focuses on inspection and interpretation. Offer multiple algorithms for global or local explanations.[63]

5.2 Debugging / Visualizing libraries

These libraries focus more on debugging or visualizing the models or data.

5.2.1 ELI5

ELI5 is a Python package that helps to debug machine learning classifiers and explain their predictions. Its focus is more on debugging the ML models by providing unified API for all of the supported frameworks and packages. Although its focus is more on displaying the weights and predictions, it also contains a way for interpreting black-box models. [75]

5.2.2 Facets

Facets are used for understanding the data we are working with. It contains two visualizations that help in understanding and analyzing datasets.

One of the visualizations is called Facets Overview and is used for a general overview of the dataset. It takes input feature data from one or more datasets and analyzes them by features and visualizes the result.

The second visualization is Facets Dive. This visualization is used to explore big amounts of data points.[8]

5.2.3 Keras Visualization Toolkit

Keras Visualization Toolkit is a toolkit for visualizing and debugging trained Keras neural net models. Provides a generalized interface for debugging or visualizing so it is easy to change the method of interpretation.[12]

5.2.4 Tf-explain

Tf-explain offers visualizations for TensorFlow 2.0 models.[19]

5.3 Libraries Comparison

In Table 5.1, we can observe which techniques the libraries are capable of. Metrics in the table refer to quantitative metrics that serve as proxies of how "good" a particular explanation is likely to be. Directly interpretable refers to §4.1, Local explanations, global explanations and data explanations are described in §3.2.

Library	Data Explanations	Directly Interpretable	Local explanations	Global Explanations	Metrics
AIX360	✓	✓	✓	✓	✓
Skater		✓	✓	✓	
ELI5			✓	✓	✓
LIME			✓		
SHAP			✓		
InterpretML		✓	✓	✓	
XAI	✓			✓	
Teller				✓	
TreeInterpreter		✓			✓
Alibi			✓	✓	✓
Facets	✓				
Keras Visualization Toolkit					✓
H2O[9]		✓	✓	✓	
DALEX[5]	✓		✓	✓	
tf-explain[19]			✓	✓	
iNNvestigate[10]			✓		

Table 5.1: Comparison of libraries based on their capabilities. Some of the libraries were taken from [26]

In table Table B.2 we can see that LIME (§4.2.1) or SHAP (§4.2.3) algorithms are fairly popular, because creators of the libraries choose to include them in their package.

In Table B.1 we can see that most of the libraries are written for Python. Note that LIME library was ported to R[12]. For SHAP library was created a wrapper in R[13].

Release years of libraries are in Table 5.2. We can see that the libraries started to appear from the year 2016, and there is a steady growth of new additions every year.

[12] `https://github.com/thomasp85/lime`

[13] `https://github.com/ModelOriented/shapper`

Library	AIX360	Skater	ELI5	LIME	SHAP	InterpretML	XAI	Teller
Release Year	2019	2017	2016	2016	2018	2019	Alpha version	No release yet

Library	TreeInterpreter	Alibi	Facets	Keras Visualization Toolkit	iNNvestigate	H2O	DALEX	
Release Year	No release yet	2019	2017	2016	2018	No release yet	2018	

Table 5.2: Table with release year of libraries (data taken from GitHub).

5.4 Library selection

We were considering which dataset to use first. We were choosing from several options like: Iris[14] , MNIST(§6.2) or Fashion-MNIST[15] dataset. Iris dataset contains only 150 instances, so this is too small and easy for nowadays standards. The MNIST dataset is described in §6.2. The fashion-MNIST dataset contains 60,000 instances and contains ten classes of clothing. Datasets differentiate with the difficulty of training the DNN (§2.3) [2]. Our experiments are meant to introduce the reader to the topic, so we choose the easier of the two: the MNIST dataset.

Since the MNIST dataset is an image classification problem, we had to choose algorithms that support images. Suitable algorithms were:
SHAP (§4.2.3), LIME (§4.2.1), TED (§4.2.4), ProfWeigth (§4.3.2). For the TED algorithm, we would have to have a special type of data, which would include explanations. Since two of the algorithms use local explanation techniques (§4.2), we could leverage this fact and compare the results of each library with each other, thus selecting LIME and SHAP.

Another benefit for chosen libraries is that both methods are relatively popular (Table B.2), so we will have better chances of comparing our results to other work.

Since we want to test the libraries' usability, we will couple one of them with AIX360 (§5.1.1). This mix is possible because AIX360 only wraps original implementation of LIME and SHAP. We will use LIME from AIX360 and SHAP from the original library.

We choose AIX360 because it is currently the biggest library with most capabilities, and according to the developers, they will be adding more algorithms into it.

[14] `https://archive.ics.uci.edu/ml/datasets/Iris`
[15] `https://github.com/zalandoresearch/fashion-mnist`

6 Use of XAI libraries

In this chapter we will use SHAP (§4.2.3) and LIME (§4.2.1) with help of AIX360 §5.1.1. We will mainly focus on simple examples with relatively common dataset (§6.2) to demonstrate the usage of chosen libraries (§5.3).

Examples will be in the form of a prototype. The prototype is written in Python and requires basic knowledge of the language. Examples are written as Jupyter Notebook and can be run in Google Colab[1], which is the easiest way of running sample code that is part of this thesis (Examples are available as an attachment and on GitHub[2].). See user manual (attachment §C) on how to run provided examples.

To design the experiments, we will follow basic ML lifecycle (Figure 6.1).

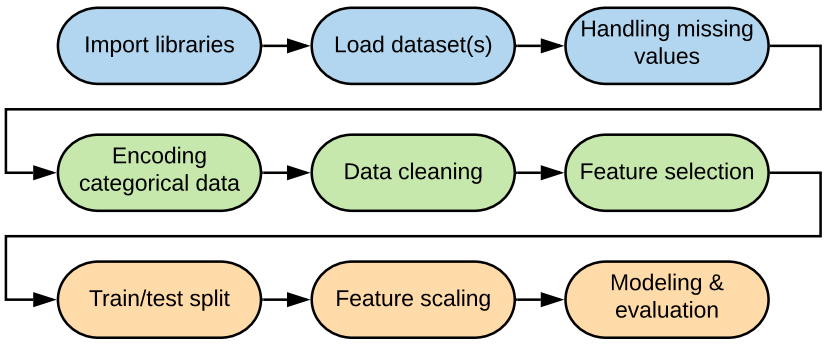


Figure 6.1: Machine learning (ML) life-cycle.

6.1 Structure of the chapter

MNIST dataset (§6.2) This section is describing the Modified National Institute of Standards and Technology (MNIST) dataset.

Common part of the examples (§6.3) Description of the common part of the examples that follow the ML lifecycle(Figure 6.1).

Using SHAP library (§6.4) Description of how to use the library.

Explaining with SHAP library (§6.4.1) Using SHAP to explain the predictions and interpret our understanding of explanation.

[1] https://colab.research.google.com/

[2] `https://github.com/fkolenak/XIA_thesis`

Using LIME library (§6.5) Description of how to use the library.

Explaining with LIME library (§6.6) Using LIME to explain the predictions and interpret our understanding of explanation.

Results comparison (§6.7) Compare our results to each other and other, similar work.

Comparison conclusion (§6.8)

6.2 MNIST handwritten digits dataset

The Modified National Institute of Standards and Technology (MNIST) database was released by [69] and contains 60,000 images of handwritten digits. An example of a few images contained is in Figure 6.2. Each image is grayscaled and is 28 pixels wide and 28 pixels high.

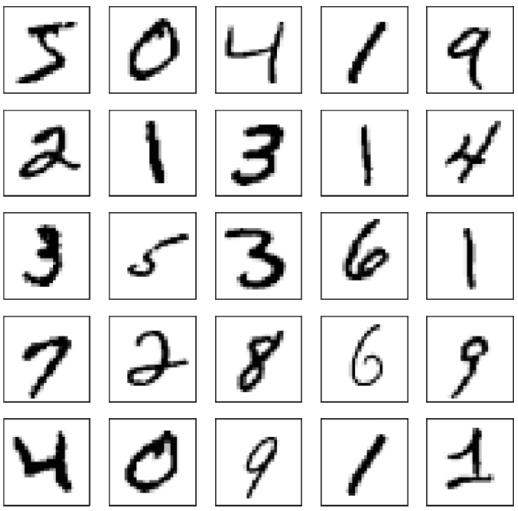


Figure 6.2: MNIST handwritten digits dataset example.

In the Figure 6.2, we can see there is much variety in the look of the digits. From the example, we can also see that some digits can cause the confusion in the sense of their shape (for instance third row, second column show digit five which is very close to the shape of digit six)

Dataset is relatively balanced (Figure 6.3, with a mean number of instances 6082 and standard deviation ±660) and does not contain any null values, so we do not have to do any handling of missing values or to clean the data.

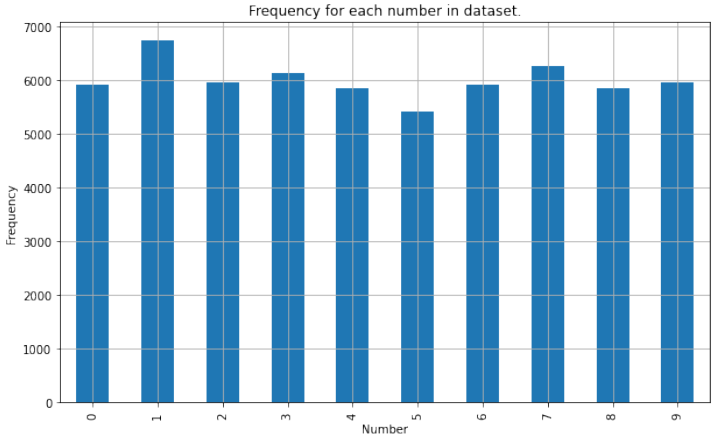


Figure 6.3: Frequency of the digits in the MNIST dataset.

6.3 General code

This chapter describes common part code implementation for both examples, i.e., experiments with SHAP (§6.4) and LIME (§6.5).

6.3.1 Importing libraries

First, we need to provide all the missing libraries to the environment we are working in. After installing all the necessary libraries, we will import packages that will be used in the program.

How to import current version of AIX360 (§5.1.1) for AIX360 and LIME experiment (§6.5) is in attachment (§D.1). List with final versions of the libraries:

- SHAP experiment §D.3
- AIX360 and LIME experiment §D.2

For our example we will be using Keras(§2.5) as our framework for creating ANN(§2.2) models, more specifically CNN(§2.4) model.

6.3.2 Loading data

The MNIST dataset is well known and broadly used, so some libraries integrated the dataset. In our case we will import MNIST from *keras.datasets* (Code Listing 6.1). The returned value is a tuple of Numpy arrays. The first two arrays are for training, and include training dataset (train) and labels (train_labels), the second set is for testing (test, test_labels).

```
# Load dataset
from keras.datasets import MNIST
# Tuple of Numpy arrays: (x_train, y_train), (x_test, y_test).
(train, train_labels), (test, test_labels) = mnist.load_data()
```

Code Listing 6.1: Loading and splitting data

6.3.3 Examine data

After importing and loading dataset (§6.3.2), we will look at some information from the data. Since we already know that the data are clean and ready to use (§6.2), we do not have to do anything in that sense.

From the data, we will extract the width and height of the image and number of classes (Code Listing 6.2). The training data have dimensions (60000, 28, 28). So, we can extract the image height and width from the shape of the training data. As for the number of classes, we will extract them from the training labels, where we count unique numbers that are within the set.

```
# save input image dimensions
img_rows = train.shape[1]
img_cols = train.shape[2]

# Get classes and number of values
value_counts = pd.value_counts(train_labels).sort_index()
num_classes = value_counts.count()
```

Code Listing 6.2: Extract dimensions and number of classes.

6.3.4 Data wrangling

To prepare data for use, we need to reshape the images to a 3-dimensional vector (height, width, color canal). We are adding the last dimension because Keras (§2.5) requires the color canal at the end.

The original 2D vector consisted of 784 values (28x28). We reshape all data to 3D matrices (28x28x1), which include canal. Our canal is equal to one because we do not have RGB colored images. For RGB images, we would reshape to 28x28x3 3D matrices, where the canal would be equal to three (R, G, and B colors).

We also perform grayscale normalization (Feature scaling) to reduce the effect of illumination's differences and speed up the convergence of CNN (because CNN is most effective when numbers are between 0-1). Division by 255 is done because the pixels, in grayscale, range from 0 - 255.

To make the vectors we will use function *reshape* (Code Listing 6.3) on the arrays with dimensions: (number of elements, width, height, number of canals).

```
# Reshape
train = train.reshape(train.shape[0], img_rows, img_cols, 1)

# Normalization
train = train/255
```

Code Listing 6.3: Reshaping arrays and grayscale normalization.

Since LIME requires as its input to have RGB images, we will reshape the data to 28x28x3 3D matrices (as mentioned before). The trick we are doing with Code Listing 6.4 is repeating one channel three times. This is often used when a model is trained on RGB pictures.

```
def to_rgb(x):
    x_rgb = np.zeros((x.shape[0], 28, 28, 3))
    for i in range(3):
        x_rgb[..., i] = x[..., 0]
    return x_rgb
```

Code Listing 6.4: Definition of a function for reshaping images.

Encoding data

Now we will need to encode labels to one-hot vector. Since we have ten classes, our one-hot vector will be 1x10. One hot vector is used to distinguish each class from every other class. The vector consists of 0s in all cells, except for a single 1 in a cell that identifies the class (normalize the weight of all classes). Example for label 4 : $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$. We will use Keras helper function *to_categorical*, which does that for us.

Cleaning and splitting data

The MNIST dataset does not require any cleaning in the data (§6.3.3).

Since we loaded the dataset from Keras (§6.3.2), the data are already split into training and testing data.

Feature scaling

To normalize we will divide whole array by 255. Example manipulation on *train array* is in Code Listing 6.3.

6.3.5 The Machine learning (ML) model

We will use CNN (§2.4), with help of Keras sequential model. The model has first layer that takes our input shape of (28, 28, 1). Output layer is dense layer with activation Softmax (§2.2.3), convolutional (§2.4.1) hidden layers use ReLu activation function (§2.2.3). Whole model structure can be seen in Figure 6.4.

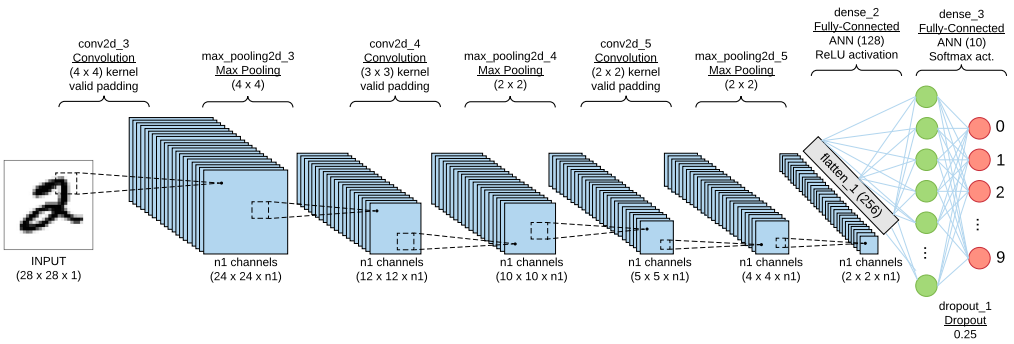


Figure 6.4: CNN model overview.

The model is created by adding layers one after another (Code Listing 6.5).

```
model.add(Conv2D(32, kernel_size = 4, activation="relu", input_shape=(
                                    img_rows,img_cols,1)))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size = 3, activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size = 2, activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
```

Code Listing 6.5: Model layers creation.

Summary of the model is in Figure 6.5. In summary, we can see each layer with parameters. This relatively simple model has 69674 parameters.

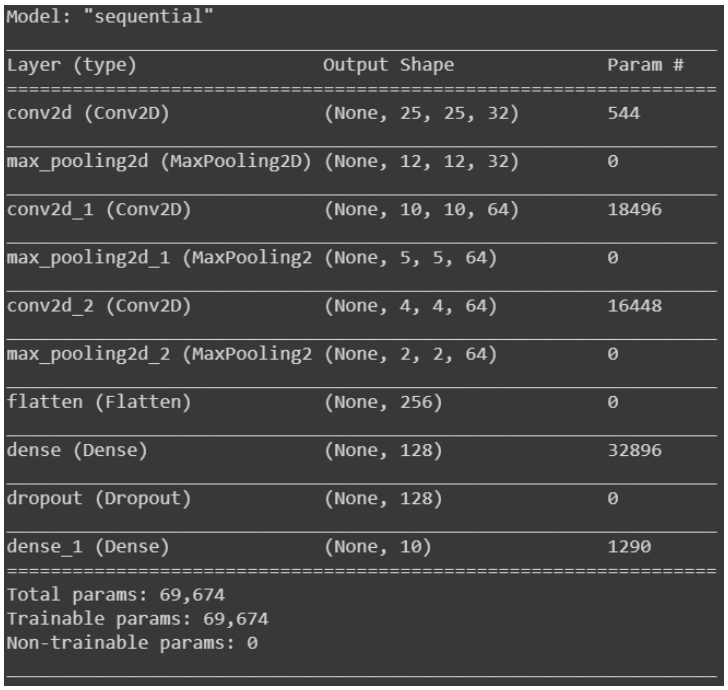


Figure 6.5: Used model summary.

6.3.6 Train the Machine learning (ML) model

After assembling our model, we will compile it (Code Listing 6.6).

Loss function (§2.2.2) Loss function is selected as *CategoricalCrossentropy*[3]. This loss function computes the crossentropy loss between the labels and predictions and is used when there are two or more label classes.

Optimizer (§2.2.4) we have chosen *Adam* optimiser. The *Adam* optimization is a stochastic gradient descent method.

Metrics (§2.2.5) From the various metrics used for ML models, the best fit is accuracy as the primary measure of model training success. This is because the dataset is balanced (§6.2), and we value the classification of positive and negative cases equally.

```
model.compile(loss='categorical_crossentropy',
      optimizer='adam',
      metrics=['accuracy'])
```

Code Listing 6.6: Compiling the model

[3] https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class

6.3.7 Model evaluation

Now that we have prepared the model, we can call the *fit* method to train it. We will set `batch_size = 128` and `epochs = 1`. Since our model is already well trained at one epoch (see Table 6.1), it is not worth spending the additional time to train it with more epochs, and we will use one epoch trained model in all of the further examples.

Epochs	Accuracy [%]	Loss [%]	Time [s]
1	97.86	6.58	42
5	99.20	2.52	210
10	99.31	2.38	420

Table 6.1: Resulting accuracy and loss based of number of epochs.

To evaluate the trained model a little bit further, we can look at the confusion matrix for 1 and 10 epochs Figure 6.6. We can see that the model had the most trouble with digits two, three, four, and seven for one epoch. For ten epochs, these digits are reduced to only six and nine.

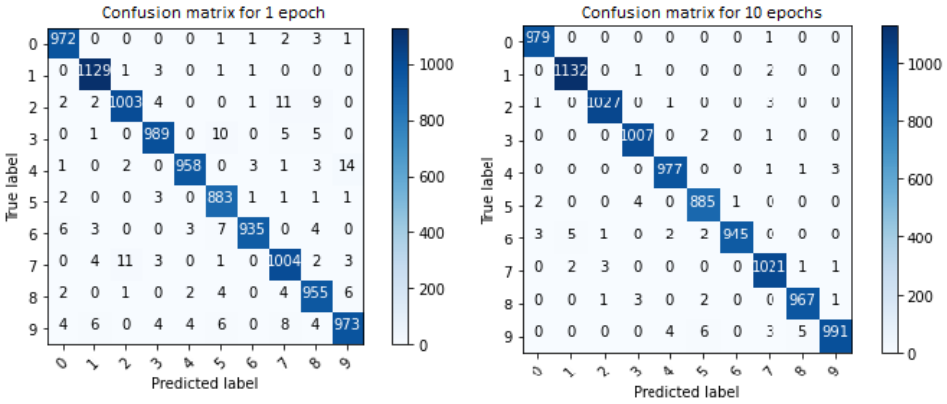


Figure 6.6: Confusion matrix for 1 and 10 epochs.

6.3.8 Test the Machine learning (ML) model

Let us take a look at the prediction that we can get from the model. Every digit in the Figure 6.7 is an example of input, which leads to the misclassification.

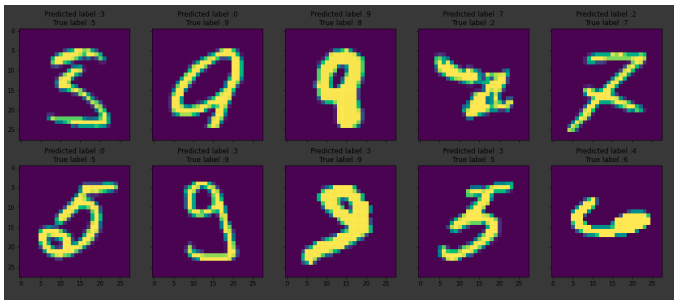


Figure 6.7: Example of digits which are missclassified by the model.

But how is that possible if some of the digits are clearly visible? Let's try to explain what model predicts by using SHAP (§6.4) and LIME (§6.5) techniques in next sections.

6.4 Using SHAP

An example is in the form of an experiment that showcases the library usage for previously trained model (§6.3.6) explanation.

We will demonstrate SHAP (§5.1.3) and its usage on MNIST dataset (§6.2). We will follow basic ML project life-cycle (Figure 6.1) and add explanation to its very end.

The common part of importing libraries, loading dataset, wrangling data, training, and evaluating model was already explained (§6.3). Now follows addition to the general code.

6.4.1 ML model explanation with SHAP

To initialize and get Shapley values, we use *DeepExplainer* (§4.2.3). In Code Listing 6.7, we can see the functions to initialize explainer and to show the explanation. We have to provide the model to the explainer as well as a subset of training data. To show the explanation, we have to provide the calculated Shapley values(§4.2.2) and instance we are explaining.

```
#Initialize
explainer = shap.DeepExplainer(model, (train[0:1000]))

# Get Shap values
shap_values = explainer.shap_values(test_errors[[i]])

# Plot
shap.image_plot(shap_values, test[i] ,index_names ,show=False)
```

Code Listing 6.7: Creating instance of explainer, getting Shapley values and visualization

The explanation's output is in the following format: the red areas in explanations are what the model expects for a given class, and blue areas are the pixels that were not what model would have expected for its label.

In the following sections, we will introduce some of the output of the explainer. We will then try to explain the explanation and comprehend the root cause for the cases.

6.4.2 SHAP on correct prediction

First lets introduce correct classification (Figure 6.8). We can see that the model fully expects this digit to look like this (red areas). The only resemblance to other digits (according to the model) is for the digit three.

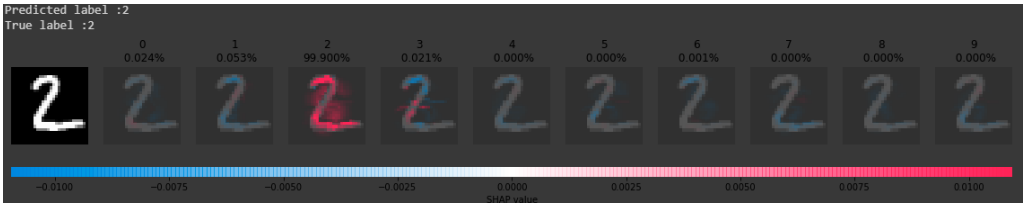


Figure 6.8: Correct classification of digit 2.

6.4.3 SHAP on incorrect predictions

In Figure 6.9 the model detected the circle with empty middle. From this observation, the lines at the end of the circle can be part of the digit 0 too, thus why it was inclined more to the 0 than 5. If we look at the correct digit, we can see that the model did not expect the line to be so close to the top part of the digit, and it did not detect the half circle that digit five is supposed to have.

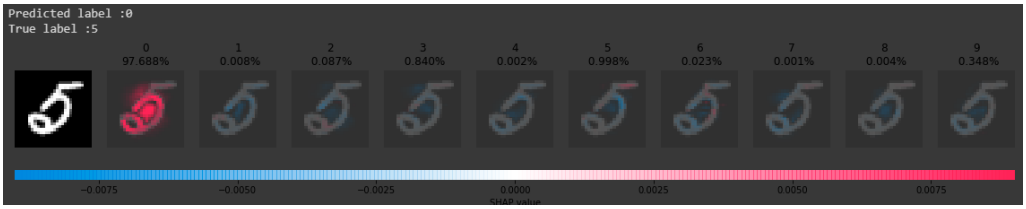


Figure 6.9: Missclassification of digit 5 as 0.

In Figure 6.10 we can observe that the shape of three is indeed in the picture. The model did not expect the connection but was still inclined to this classification. From this example, we can see that some digits can have a

different orientation. To improve the accuracy, we could introduce some image distortion like rotations or zoom and add it as a layer to the model (§7.1).

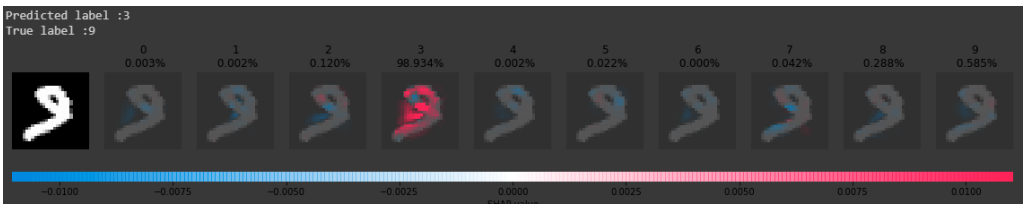


Figure 6.10: Missclassification of digit 9 as 3.

In Figure 6.11 we can only assume that the model did not detect the digit nine because the training set did not have any or not many examples with this specific writing style (the line at the bottom).

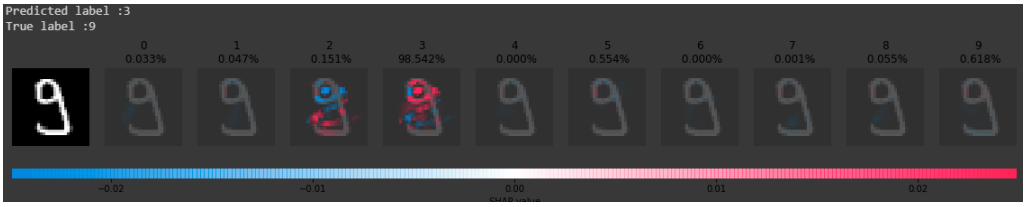


Figure 6.11: Missclassification of digit 9 as 3.

6.5 Using LIME with AIX360

An example is in the form of an experiment that showcases the usage of the library as well as using the library for explanations. In this example we will use library AI Explainability 360 (AIX360) (§5.1.1) and its classes for Local Interpretable Model—Agnostic Explanations (LIME) (§4.2.1) explanation approach.

We have covered the process of how the model is built in common code section (§6.3). So, we will not explain it in detail again. For future comparison we will also use MNIST (§6.2) dataset. The ML life-cycle from previous example (§6.4.1) will be very similar, with the exception that instead of using SHAP we will do the explanation with LIME at this time.

6.6 ML model explanation with LIME

To initialize and get explanations we use object from AIX360 *LimeImageExplainer*. In Code Listing 6.8, we can see the functions to initialize explainer

and to show the explanation. We have to provide the model to the explainer as well as a subset of training data. To show the explanation, we have to provide an instance we are explaining, and models function to get the probability, for instance.

```
#Initialize
limeExplainer = LimeImageExplainer()

# Get explanation
explanation = limeExplainer.explain_instance(test_errors_rgb[i], model.
                                             predict)

# Get image and mask and show result
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
                                            positive_only=True, num_features=10,
                                            hide_rest=True)

plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

Code Listing 6.8: LIME creating explanation model and showing explanation.

Further image examples are always in a format: original image, an image with superpixels highlighted, an image with the 'pros and cons'. Pros and cons mean that parts of the image are highlighted (pros are green and cons are red). In the examples there is also text with *Top predictions: [Array]*. This array is displaying the top 5 predictions. So the second digit has the second-best probability and so on.

6.6.1 LIME on correct predictions

In the examples Figure 6.12 and Figure 6.13 we can see that LIME divided the image into superpixels. Each superpixel can be seen as a yellow line in the second and third image. With the first example we can see that the digit nine has to have empty circle in the middle. Also notice how small the pros part is in the images.

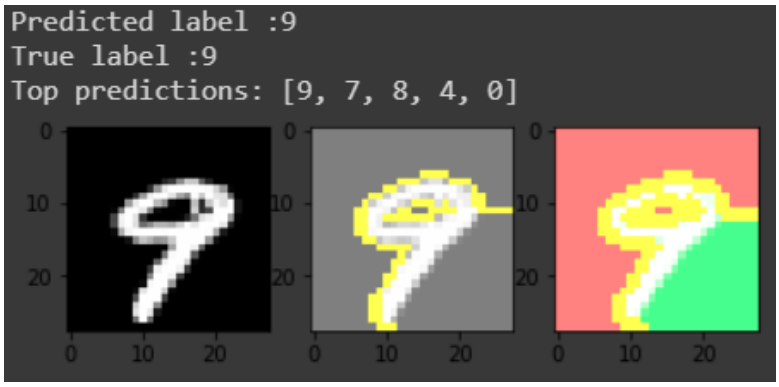


Figure 6.12: LIME explanation - correctly classified digit 9.

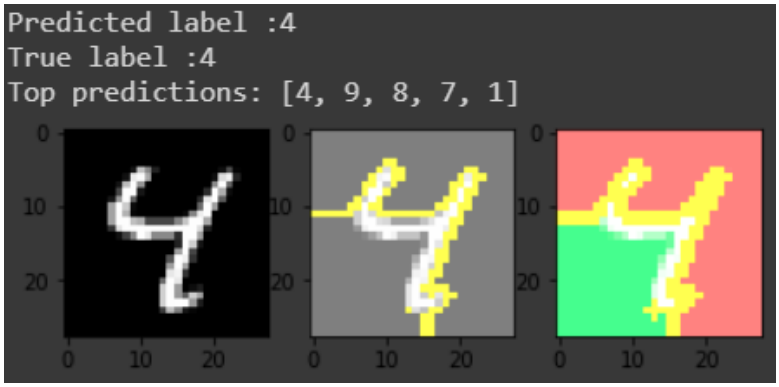


Figure 6.13: LIME explanation - correctly classified digit 4.

Example Figure 6.14 is a special case, because LIME explanation did not seem to pick up any superpixels or given any feasible interpretation. This could be caused by the dimensions of the image, since 784 pixels is quite small picture or it could be caused by the model itself where LIMEs effort have no change in the prediction. To get better results from LIME, we tried to alter some input parameters (In Code Listing 6.8 for a function *get_image_and_mask* of explanation object) but the quality of the explanations did not increase.

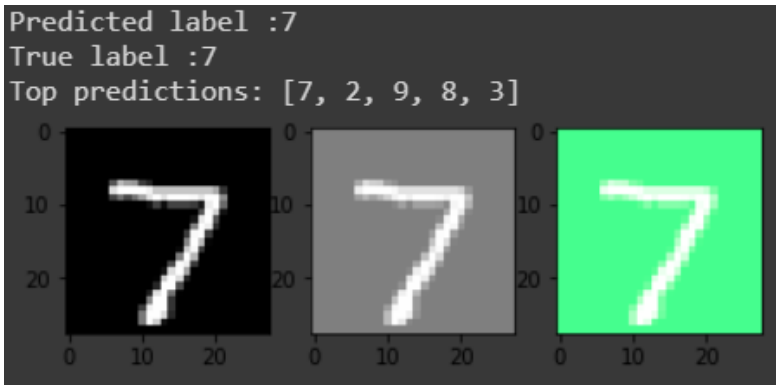


Figure 6.14: LIME explanation - correctly classified digit 7, but no explanation given.

6.6.2 LIME on incorrect predictions

If we take a look at some explanations that try to explain incorrect prediction Figure 6.15 and Figure 6.16 we will notice that the pros part of the explanation is much bigger than in the previous, correct, examples. Looking at the images, the explanation only vaguely follows the shape and then includes the whole digit. Unfortunately, the superpixels were formed only for the outer line and did not follow any actual shape.

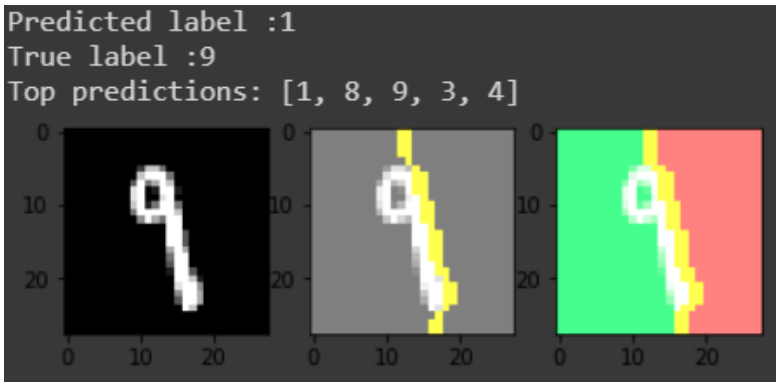


Figure 6.15: LIME explanation - incorrect explanation.

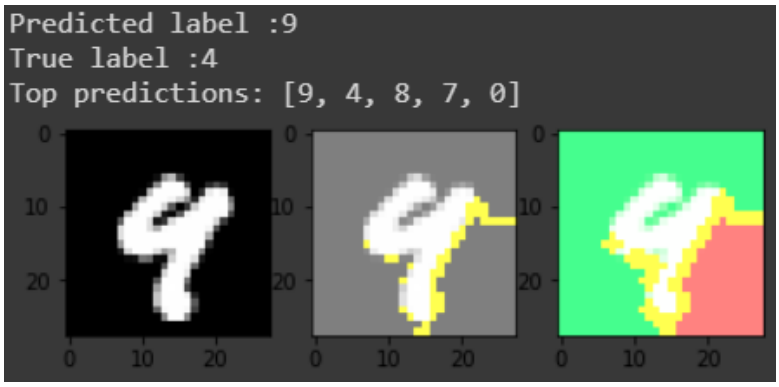


Figure 6.16: LIME explanation - incorrect explanation.

6.6.3 LIME on incorrect predictions with heatmap

In the explanation, using LIME can be helpful to generate heatmap (Figure 6.17) that showcases the main focus points in the image. From the image, it is more apparent that the model is also looking at the shape and detects the inner circle. However, other than that, we see no significant explanation of the misclassified prediction.

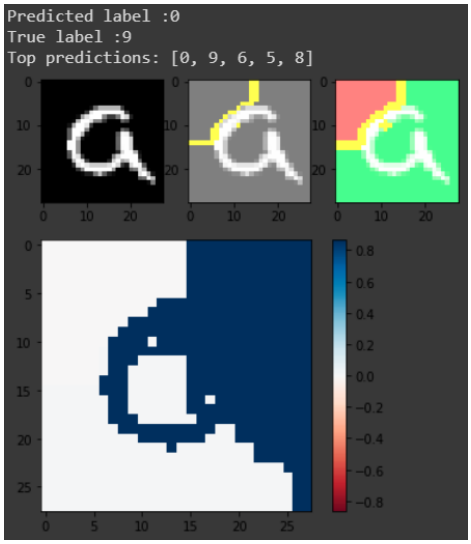


Figure 6.17: LIME explanation - Explanation with addition of heatmap.

If we generate new image (Figure 6.18) with heatmap for the example in Figure 6.16 we can see more clearly what matters to the model, but still no useful explanation.

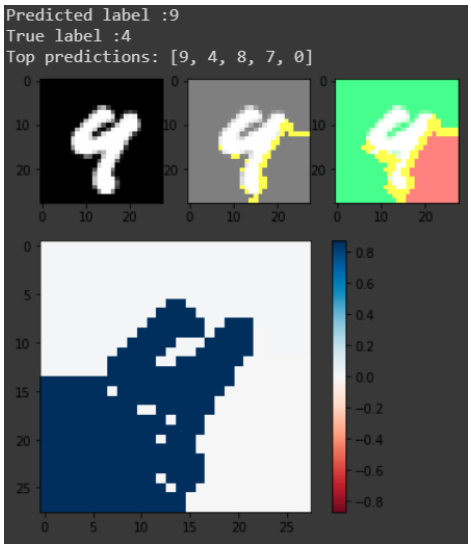


Figure 6.18: LIME explanation - Explanation with addition of heatmap.

6.7 Explanation comparison

Since the used MNIST dataset (§6.2) is fairly popular amongst researchers, we will try to compare our results with other works (§6.7.2, §6.7.2 and §6.7.2).

6.7.1 Our work results comparison

We will first compare our results for the same instance using explanation with LIME (§6.5) and SHAP (§6.4).

In Figure 6.21 it was incorrectly predicted that the label is 4. Comparing results, we can see meaningful explanation only for SHAP algorithm. Same phenomena is visible in the Figure 6.20, where LIME does not provide any explanation even in the heatmap.

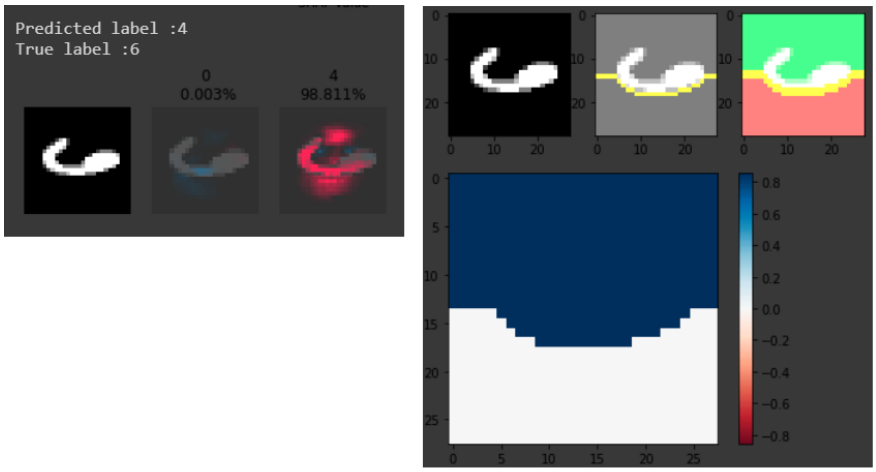


Figure 6.19: Comparison of our results. SHAP at the left and LIME at the right.

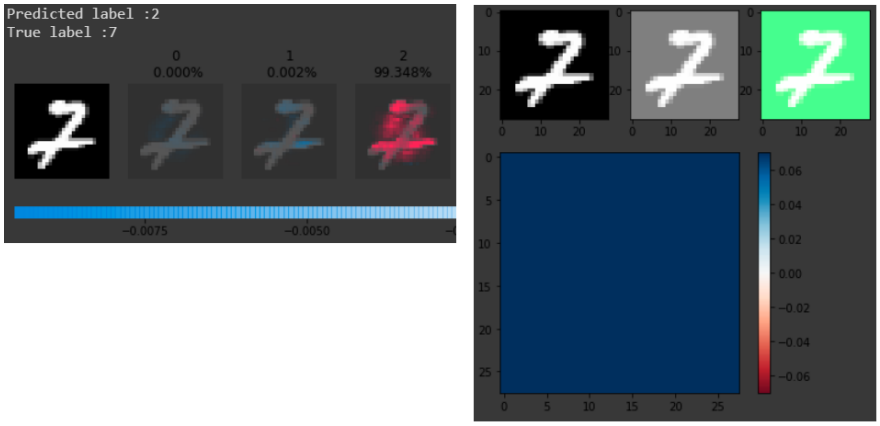


Figure 6.20: Comparison of our results. SHAP at the left and LIME at the right.

6.7.2 Lundberg and Lee [70] results comparison

Lets take first example Figure 6.21 from [70]. If we compare it to our results, we can see that our approach for SHAP(§6.4.1) was correct, because we have similar-looking results. In their example, they comment on the result, just explaining the red and blue areas. The example includes a comparison to other classes and a masked version where they removed the unwanted parts for class 3 and displayed the result (Masked column). Unfortunately, they do not mention how they obtained results for LIME, because, for their case, it seems to work fine.

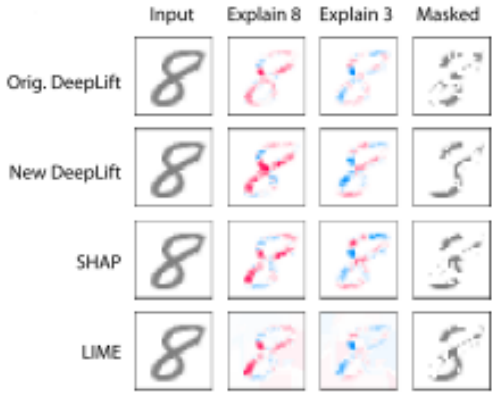


Figure 6.21: Image explanation results [70].

6.7.3 Kanerva [59] results comparison

Other Figure 6.22 show that our result with LIME on this dataset seems to be the same. They also got no explanation for some cases (Figure 6.14), or wrong explanations like in Figure 6.15. In the thesis, they also mention that they tried to change some parameters without results.

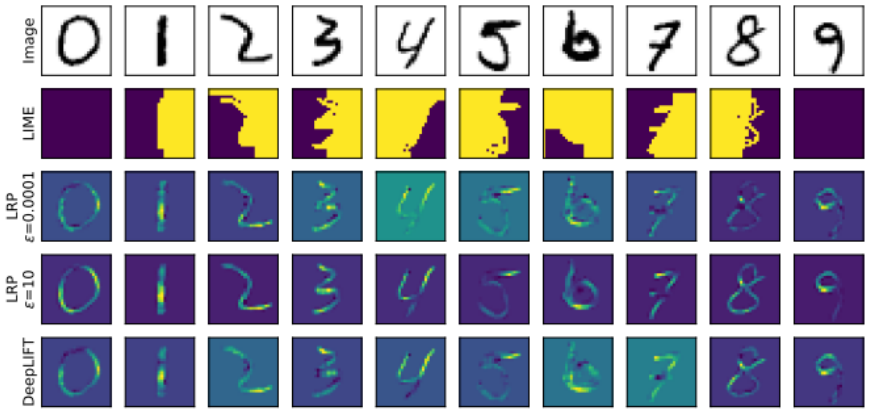
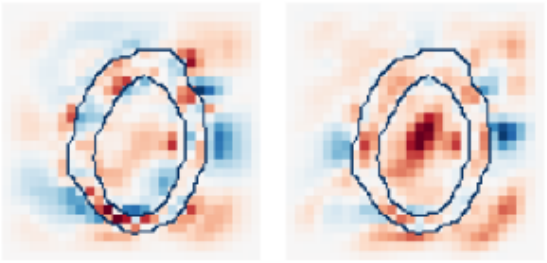


Figure 6.22: Image with digits prediction explanation results [59].

6.7.4 Ancona et al. [24] results comparison

From the results of this report, we can see that the explanation output of SHAP library is similarly looking to other libraries (Example in Figure 6.23). In those examples, we can see that the model's focus point is very similar to SHAPs explanation.



(a) Occlusion-1 (b) Integrated Gradients

Figure 6.23: Example of output by Oclusion [109] and Integrated gradients methods [102]. Image courtesy of [24].

6.7.5 Ilamanov [57] approach

The [57] presents an interesting approach where they interpret the model at each layer and look at the activation at that layer. After printing the output, they can interpret what each layer does (e.g., horizontal line detection).

6.8 Comparison conclusion

From our experiments (§6.4 and §6.5), we can see that explaining the results of a model is one thing, and another is understanding the explanation itself. It depends on the reviewers' view and their interpretation of the results.

Comparing our work and other people's work (§6.7), we can see that the algorithm which is chosen for the explanation can produce different results (§6.7.2 and §6.7.3), depending on the architecture of the whole system.

From [**?**] and [**?**], we can see other algorithms that can perform well on the same dataset and model. Thus the algorithm choice plays a big part in the success of explanation.

7 Conclusion

In this thesis, we have been introduced to XAI(§3) problematic and explained why it is essential to be able to interpret AIs decisions. We have given an introduction to the AI(§2), which is the main point of interest in XAI. We have described the approaches for explanation and presented some XAI algorithms.

We have searched for XAI libraries and found out that many are still in the beginning stage(§5.3) of development. By far, the most used libraries were LIME and SHAP.

In the experiments, we have tried LIME and SHAP with the MNIST dataset. We have used one of the biggest library AIX360 that includes LIME. We used the original SHAP library that was released with its research paper. With the AIX360, we have tested how simple it is to use this library. With the used libraries, we verified the simplicity of explaining and understanding the behavior of the model.

After comparing other works that use other explanation algorithms for the dataset used, the SHAP had the best interpretability of its explanation output. LIME, in our case, did not provide any meaningful explanation.

Experiments revealed some information about the dataset used, where we were able to spot some irregularities(§6.4) within the dataset. Experiments also pointed out that some of the numbers were written differently in the testing set (Figure 6.11), so the model did not predict those instances correctly.

7.1 Future work

With the explanation method, we found that within the dataset are some numbers that have the desired shape but are slightly angled (§6.4.3 Figure 6.10). We could use data augmentation (Figure 7.1) [107] to further improve the model's accuracy. From the data augmentations available, the rotation would be most helpful. Other techniques mostly apply to colored images. With the usage of such technique, we will also increase the training dataset, which is another added benefit.

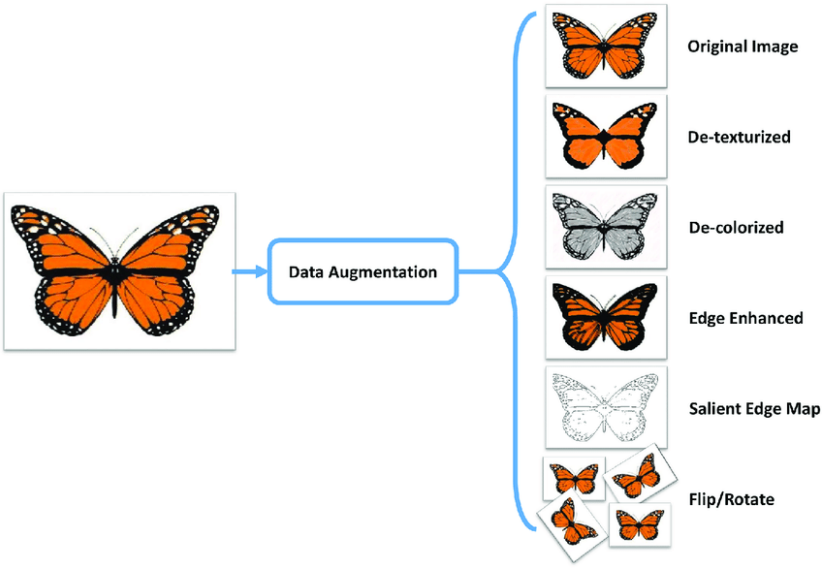


Figure 7.1: Example data augmentation on a picture of a butterfly. For our set of images, the rotation is most relevant [13].

Another improvement to models accuracy could be achieved with the usage of Automated machine learning (also referred to as AutoML) tools. AutoML method aims to automate the process of ML model development. It automatically optimizes hyperparameters of the model and selects the best one based on metrics selected.

Today's XAI is in a state where we have to sacrifice accuracy for explainability(§4). In the future, the aim will be to create more explainable models (Figure 7.2) and, at the same time, maintain high performance. XAI will also enable the trust of the AI systems by enabling the users to understand its predictions[47].

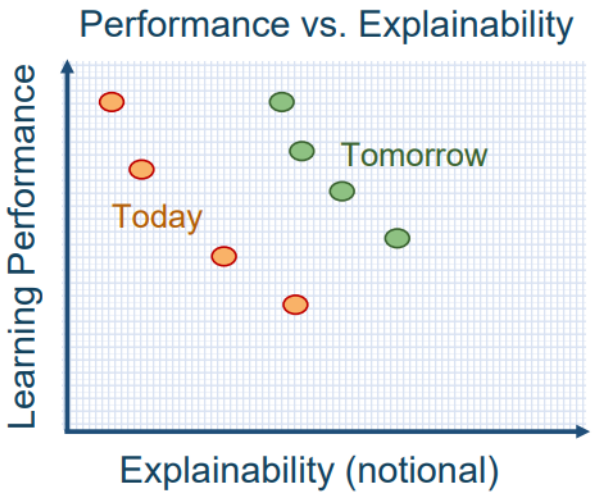


Figure 7.2: Darpas vision on the future of XAI [47].

Bibliography

[1] AlphaGo | DeepMind. URL https://deepmind.com/research/case-studies/alphago-the-story-so-far. Last visited 2020-07-31.

[2] Benchmark dashboard. URL http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#. Last visited 2020-08-09.

[3] Top scientists call for caution over artificial intelligence - Telegraph. URL https://www.telegraph.co.uk/technology/news/11342200/Top-scientists-call-for-caution-over-artificial-intelligence.html. Last visited 2020-08-09.

[4] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-\networks-the-eli5-way-3bd2b1164a53. Last visited 2020-08-05.

[5] ModelOriented/DALEX: moDel Agnostic Language for Exploration and eXplanation. URL https://github.com/ModelOriented/DALEX. Last visited 2020-08-05.

[6] IBM100 - Deep Blue. URL https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. Last visited 2020-07-31.

[7] The AI-RFX Procurement Framework for AI Machine Learning Systems. URL https://ethical.institute/xai.html. Last visited 2020-07-31.

[8] Facets - Visualizations for ML datasets. URL https://pair-code.github.io/facets/. Last visited 2020-07-31.

[9] h2oai/mli-resources: H2O.ai Machine Learning Interpretability Resources. URL https://github.com/h2oai/mli-resources. Last visited 2020-08-05.

[10] albermax/innvestigate: A toolbox to iNNvestigate neural networks' predictions! URL https://github.com/albermax/innvestigate. Last visited 2020-08-05.

[11] Keras: the Python deep learning API, . URL https://keras.io/. Last visited 2020-08-05.

[12] GitHub - raghakot/keras-vis: Neural network visualization toolkit for keras, . URL https://github.com/raghakot/keras-vis. Last visited 2020-07-31.

[13] Data Augmentation Increases Accuracy of your model — But how ? | by sourav kumar | Secure and Private AI Writing Challenge | Medium. URL https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-\your-model-but-how-aa1913468722. Last visited 2020-08-09.

[14] Moral Machine. URL https://www.moralmachine.net/. Last visited 2020-08-09.

[15] PyTorch. URL https://pytorch.org/. Last visited 2020-08-05.

[16] Overview — skater 0 documentation. URL https://oracle.github.io/Skater/overview.html. Last visited 2020-07-30.

[17] scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation. URL https://scikit-learn.org/stable/index.html. Last visited 2020-08-06.

[18] TensorFlow. URL https://www.tensorflow.org/. Last visited 2020-08-05.

[19] Available Methods — tf-explain documentation. URL https://tf-explain.readthedocs.io/en/latest/methods.html. Last visited 2020-07-31.

[20] GitHub - andosa/treeinterpreter. URL https://github.com/andosa/treeinterpreter. Last visited 2020-07-31.

[21] Igor Aleksander. Artificial neuroconsciousness an update. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 930, pages 565–581. Springer Verlag, 1995. ISBN 3540594973. doi: 10.1007/3-540-59497-3_224.

[22] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2014. ISBN 9780262028189.

[23] Rayan Alshamrani and Xiaogang Ma. Deep Learning. In *Encyclopedia of Big Data*, pages 1–5. Springer International Publishing, Cham, 2019. doi: 10.1007/978-3-319-32001-4_533-1. URL http://link.springer.com/10.1007/978-3-319-32001-4_533-1.

[24] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. TOWARDS BETTER UNDERSTANDING OF GRADIENT-BASED ATTRIBUTION METHODS FOR DEEP NEURAL NETWORKS. Technical report, 2018.

[25] Stevo and others Bozinovski. A self-learning system using secondary reinforcement. *Cybernetics and Systems Research*, pages 397—-402, 1982.

[26] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. sep 2019. URL http://arxiv.org/abs/1909.03012.

[27] Edmond Awad, Sohan Dsouza, Richard Kim, Jonathan Schulz, Joseph Henrich, Azim Shariff, Jean François Bonnefon, and Iyad Rahwan. The Moral Machine experiment. *Nature*, 563(7729):59–64, nov 2018. ISSN 14764687. doi: 10.1038/s41586-018-0637-6. URL https://doi.org/10.1038/s41586-018-0637-6.

[28] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I. Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang. An Introduction to Machine Learning. *Clinical Pharmacology and Therapeutics*, 107(4):871–885, apr 2020. ISSN 15326535. doi: 10.1002/cpt.1796.

[29] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, jun 2020. ISSN 1566-2535. doi: 10.1016/J.INFFUS.2019.12.012. URL https://www.sciencedirect.com/science/article/pii/S1566253519308103#bib0016.

[30] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, 35(8):1798–1828, jun 2013. ISSN 01628828. doi: 10.1109/TPAMI.2013.50. URL http://arxiv.org/abs/1206.5538.

[31] Christopher Michael Bishop. Pattern Recognition and Machine Learning. *Journal of Electronic Imaging*, 16(4):049901, jan 2007. ISSN 1017-9909. doi: 10.1117/1.2819119.

[32] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. dec 2014. URL http://arxiv.org/abs/1412.3555.

[33] Miruna A Clinciu and Helen F Hastie. Proceedings of the 1st Workshop on Interactive Natural Language Technology for Explainable Artificial Intelligence (NL4XAI 2019). Technical report.

[34] Daniel Crevier. *AI : the tumultuous history of the search for artificial intelligence*. Basic Books, 1993. ISBN 0465029973.

[35] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, dec 1989. ISSN 09324194. doi: 10.1007/BF02551274. URL https://link.springer.com/article/10.1007/BF02551274.

[36] Sanjeeb Dash, Oktay Günlük, and Dennis Wei. Boolean Decision Rules via Column Generation. Technical report, 2018.

[37] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. Technical report, 2018.

[38] Amit Dhurandhar, Karthikeyan Shanmugam, Ronny Luss, and Peder Olsen. Improving Simple Models with Confidence Profiles. Technical report, 2018.

[39] Nicholas Diakopoulos and Michael Koliska. Algorithmic Transparency in the News Media. *Digital Journalism*, 5(7):809–828, aug 2017. ISSN 2167082X. doi: 10.1080/21670811.2016.1208053. URL https://www.tandfonline.com/doi/abs/10.1080/21670811.2016.1208053.

[40] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. feb 2017. URL http://arxiv.org/abs/1702.08608.

[41] Jeffreyl Elman. Finding Structure in Time. Technical report, 1990.

[42] D. A. Ferrucci. Introduction to "This is Watson", may 2012. ISSN 00188646.

[43] Xiao Gang Su. *Linear regression analysis: Theory and computing*. World Scientific Publishing Co., jan 2009. ISBN 9789812834119. doi: 10.1142/6986. URL https://www.worldscientific.com/worldscibooks/10.1142/6986https://books.google.cz/books?id=MjNv6rGv8NIC&pg=PA1&redir_esc=y#v=onepage&q&f=false.

[44] Adam E. Gaweda, Mehmet K. Muezzinoglu, George R. Aronoff, Alfred A. Jacobs, Jacek M. Zurada, and Michael E. Brier. Individualization of pharmacological anemia management using reinforcement learning. In *Neural Networks*, volume 18, pages 826–834. Pergamon, jul 2005. doi: 10.1016/j.neunet.2005.06.020.

[45] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 3, pages 2672–2680. Neural information processing systems foundation, 2014.

[46] Steven L. Gortmaker, David W. Hosmer, and Stanley Lemeshow. Applied Logistic Regression. *Contemporary Sociology*, 23(1):159, jan 1994. ISSN 00943061. doi: 10.2307/2074954.

[47] David Gunning. Explainable Artificial Intelligence (XAI). Technical report, 2017. URL https://www.darpa.mil/attachments/XAIProgramUpdate.pdf.

[48] David Gunning and David W Aha. DARPA's Explainable Artificial Intelligence Program Deep Learning and Security. *AI Magazine: Deep Learning and Security*, 2019. ISSN 0738-4602.

[49] David Gunning, Filip Karlo Dosilovic, Mario Brcic, Nikica Hlupic, Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, Francisco Herrera, Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, Amit Dhurandhar, Karthikeyan Shanmugam, Ronny Luss, Peder Olsen, Tim Miller, Avi Rosenfeld, Ariella Richardson, Bryce Goodman, Seth Flaxman, Xiaocong Cui, Jung min Lee, J. Po-An Hsieh, David Gunning, and David W Aha. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58 (3):82–115, jun 2018. ISSN 1566-2535. doi: 10.1016/J.INFFUS.2019.12.012. URL http://arxiv.org/abs/1904.08123http:

//arxiv.org/abs/1706.07269https://www.sciencedirect.com/science/article/pii/S1566253519308103#bib0016https://ieeexplore.ieee.org/document/8400040/http://listverse.com/.

[50] Karthik S. Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient Data Representation by Selecting Prototypes with Importance Weights. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2019-Novem:260–269, jul 2017. URL http://arxiv.org/abs/1707.01212.

[51] Kjell Hausken and Matthias Mohr. The value of a player in n-person games. *Social Choice and Welfare*, 18(3):465–483, 2001. ISSN 01761714. doi: 10.1007/s003550000070.

[52] Michael Hind, Dennis Wei, Murray Campbell, Noel C.F. Codella, Amit Dhurandhar, Aleksandra Mojsilović, Karthikeyan Natesan Ramamurthy, and Kush R. Varshney. TED: Teaching AI to explain its decisions. In *AIES 2019 - Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 123–129, New York, NY, USA, jan 2019. Association for Computing Machinery, Inc. ISBN 9781450363242. doi: 10.1145/3306618.3314273. URL https://dl.acm.org/doi/10.1145/3306618.3314273.

[53] Geoffrey Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009. ISSN 1941-6016. doi: 10.4249/scholarpedia.5947.

[54] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

[55] Chia-Ling Huang, Yan-Chih Shih, Chyh-Ming Lai, Vera Yuk Ying Chung, Wen-Bo Zhu, Wei-Chang Yeh, and Xiangjian He. Optimization of a Convolutional Neural Network Using a Hybrid Algorithm. In *2019 International Joint Conference on Neural Networks (IJCNN)*, volume 2019-July, pages 1–8. IEEE, jul 2019. ISBN 978-1-7281-1985-4. doi: 10.1109/IJCNN.2019.8852231. URL https://ieeexplore.ieee.org/document/8852231/.

[56] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. 2016.

[57] Nazar Ilamanov. Explainable MNIST classification: dissection of a ConvNet | by Nazar Ilamanov | Jul, 2020 | Towards Data Science, 2020. URL https://towardsdatascience.com/explainable-mnist-classification-dissection\-of-a-convnet-f32910d52842. Last visited 2020-08-06.

[58] Bogumił Kamiński, Michał Jakubczyk, and Przemysław Szufel. A framework for sensitivity analysis of decision trees. *Central European Journal of Operations Research*, 26(1):135–159, mar 2018. ISSN 16139178. doi: 10.1007/s10100-017-0479-6. URL https://doi.org/10.1007/s10100-017-0479-6.

[59] Olli Kanerva. Evaluating explainable AI models for convolutional neural networks with proxy tasks. Technical report, 2019. URL https://pdfs.semanticscholar.org/d910/62a3e13ee034af6807e1819a9ca3051daf13.pdf.

[60] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1746–1751. Association for Computational Linguistics (ACL), 2014. ISBN 9781937284961. doi: 10.3115/v1/d14-1181.

[61] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015. URL https://arxiv.org/abs/1412.6980v9.

[62] Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. Semi-Supervised Learning with Deep Generative Models. *Advances in Neural Information Processing Systems*, 4(January):3581–3589, jun 2014. URL http://arxiv.org/abs/1406.5298.

[63] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. Indices and tables — Alibi 0.5.2dev documentation. URL https://docs.seldon.io/projects/alibi/en/latest/. Last visited 2020-07-31.

[64] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, feb 1991. ISSN 15475905. doi: 10.1002/aic.690370209. URL https://aiche.onlinelibrary.wiley.com/doi/full/10.1002/aic.690370209https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209https://aiche.onlinelibrary.wiley.com/doi/10.1002/aic.690370209.

[65] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations. *6th International Conference on Learning Representations,*

ICLR 2018 - Conference Track Proceedings, nov 2017. URL http://arxiv.org/abs/1711.00848.

[66] Christopher Kuner, Fred Cate, Orla Lynskey, Christopher Millard, Nora Ni Loideain, and Dan Svantesson. Meaningful information and the right to explanation . *International Data Privacy Law*, 7(2):73–75, nov 2017. doi: 10.1093/IDPL. URL https://cyber.harvard.edu/publications/2018/03/BigDataPrivacy.

[67] Pat Langley. The changing science of machine learning, mar 2011. ISSN 08856125. URL http://archive.ics.uci.edu/ml/.

[68] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 8595–8598, dec 2011. URL http://arxiv.org/abs/1112.6209.

[69] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998. ISSN 00189219. doi: 10.1109/5.726791.

[70] Scott Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 2017-Decem:4766–4775, may 2017. URL http://arxiv.org/abs/1705.07874.

[71] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, jan 2020. doi: 10.1038/s42256-019-0138-9. URL https://arxiv.org/abs/1905.04610.

[72] Ronny Luss, Pin-Yu Chen, Amit Dhurandhar, Prasanna Sattigeri, Yunfeng Zhang, Karthikeyan Shanmugam, and Chun-Chen Tu. Generating Contrastive Explanations with Monotonic Attribute Functions. may 2019. URL http://arxiv.org/abs/1905.12698.

[73] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5 (4):115–133, dec 1943. ISSN 00074985. doi: 10.1007/BF02478259. URL https://link.springer.com/article/10.1007/BF02478259.

[74] Charles E. Metz. Basic principles of ROC analysis. *Seminars in Nuclear Medicine*, 8(4):283–298, 1978. ISSN 00012998. doi: 10.1016/S0001-2998(78)80014-2. URL https://pubmed.ncbi.nlm.nih.gov/112681/.

[75] Korobov Mikhail and Lopuhin Konstantin. Welcome to ELI5's documentation! — ELI5 0.9.0 documentation. URL https://eli5.readthedocs.io/en/latest/index.html. Last visited 2020-07-29.

[76] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences, feb 2019. ISSN 00043702. URL http://arxiv.org/abs/1706.07269.

[77] Christoph Molnar. *Interpretable Machine Learning*. 2019. URL https://christophm.github.io/interpretable-ml-book/.

[78] T. Moudiki. GitHub - thierrymoudiki/teller: Model-agnostic Machine Learning explainability. URL https://github.com/thierrymoudiki/teller. Last visited 2020-07-31.

[79] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. Technical report, 2010.

[80] Andrew Ng. Extract Data Conference | SlideShare. URL https://www.slideshare.net/ExtractConf. Last visited 2020-08-09.

[81] Tong Niu and Mohit Bansal. Adversarial Over-Sensitivity and Over-Stability Strategies for Dialogue Models. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 486–496, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-1047. URL https://www.aclweb.org/anthology/K18-1047.

[82] Ritesh Noothigattu, Snehalkumar 'Neil' S. Gaikwad, Edmond Awad, Sohan Dsouza, Iyad Rahwan, Pradeep Ravikumar, and Ariel D. Procaccia. A Voting-Based System for Ethical Decision Making. sep 2017. URL http://arxiv.org/abs/1709.06692.

[83] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. InterpretML: A Unified Framework for Machine Learning Interpretability. sep 2019. URL http://arxiv.org/abs/1909.09223.

[84] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. nov 2015. URL http://arxiv.org/abs/1511.08458.

[85] Ravindra Parmar. Common Loss functions in machine learning | by Ravindra Parmar | Towards Data Science. URL https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. Last visited 2020-08-08.

[86] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction*. SPIE, sep 2009. doi: 10.1117/3.633187.

[87] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 15730565. doi: 10.1023/A:1022643204877.

[88] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, sep 1987. ISSN 00207373. doi: 10.1016/S0020-7373(87)80053-6.

[89] Shiv Ram Dubey, Soumendu Chakraborty, Swalpa Kumar Roy, Student Member, Snehasis Mukherjee, Satish Kumar Singh, Senior Member, and Bidyut Baran Chaudhuri. diffGrad: An Optimization Method for Convolutional Neural Networks. Technical report. URL https://github.com/shivram1987/diffGrad.

[90] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Augu, pages 1135–1144. Association for Computing Machinery, aug 2016. ISBN 9781450342322. doi: 10.1145/2939672.2939778.

[91] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are Loss Functions All the Same? *Neural Computation*, 16(5):1063–1076, may 2004. ISSN 08997667. doi: 10.1162/089976604773135104. URL https://www.mitpressjournals.org/doi/10.1162/089976604773135104.

[92] Sebastian Ruder, Matthew E Peters, Swabha Swayamdipta, and Thomas Wolf. Transfer Learning in Natural Language Processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-5004. URL https://www.aclweb.org/anthology/N19-5004.

[93] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic Routing Between Capsules. *Advances in Neural Information Processing Systems*,

2017-Decem:3857–3867, oct 2017. URL http://arxiv.org/abs/1710.09829.

[94] Mansour Saffar. An Introduction to XAI! Towards Trusting Your ML Models! URL https://www.slideshare.net/MansourSaffarMehrjar/an-introduction-to-xai-towards-trusting-your-ml-models. Last visited 2020-08-09.

[95] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Toward Interpretable Machine Learning: Transparent Deep Neural Networks and Beyond. Technical report, 2020.

[96] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1-2):207–219, 2000. ISSN 00188646. doi: 10.1147/rd.441.0206.

[97] Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117, apr 2014. doi: 10.1016/j.neunet.2014.09.003. URL http://arxiv.org/abs/1404.7828http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[98] A. Carlisle Scott, William J. Clancey, Randall Davis, and Edward H. Shortliffe. Explanation capabilities of production-based consultation systems. *American Journal of Computational Linguistics*, 1977. URL https://www.academia.edu/11212222/EXPLANATION_CAPABILITIES_Of_PRODUCTION_BASED_CONSULTATION_SYSTEMS.

[99] Avanti Shrikumar, Peyton Greenside, Anna Y Shcherbina, and Anshul Kundaje. Not Just A Black Box: Learning Important Features Through Propagating Activation Differences. Technical report, 2016. URL https://arxiv.org/.

[100] Stephen V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, oct 1997. ISSN 00344257. doi: 10.1016/S0034-4257(97)00083-7.

[101] Susan Stuart. Alvin I. Goldman, simulating minds: The philosophy, psychology and neuroscience of mindreading. *Minds and Machines*, 19(2):279–282, may 2009. ISSN 09246495. doi: 10.1007/s11023-009-9142-x. URL https://link.springer.com/article/10.1007/s11023-009-9142-x.

[102] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. Technical report, jul 2017. URL http://proceedings.mlr.press/v70/sundararajan17a.html.

[103] William R. Swartout. Explaining and Justifying Expert Consulting Programs. pages 254–271. 1981. doi: 10.1007/978-1-4612-5108-8_15. URL http://link.springer.com/10.1007/978-1-4612-5108-8_15.

[104] Andreas M. Tillmann. On the computational intractability of exact and approximate dictionary learning. *IEEE Signal Processing Letters*, 22(1): 45–49, 2015. ISSN 10709908. doi: 10.1109/LSP.2014.2345761.

[105] A M Turing. COMPUTING MACHINERY AND INTELLIGENCE. Technical report, 1950. URL https://www.csee.umbc.edu/courses/471/papers/turing.pdf.

[106] Fjodor Van Veen. The Neural Network Zoo - The Asimov Institute. URL https://www.asimovinstitute.org/neural-network-zoo/. Last visited 2020-08-08.

[107] Jason Wang and Luis Perez. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Technical report.

[108] Dennis Wei, Sanjeeb Dash, Tian Gao, and Oktay G ¨ Unï Uk. Generalized Linear Rule Models. Technical report, may 2019. URL http://proceedings.mlr.press/v97/wei19a.html.

[109] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833. Springer Verlag, nov 2014. ISBN 9783319105895. doi: 10.1007/978-3-319-10590-1_53. URL https://arxiv.org/abs/1311.2901v3.

[110] Czako Zoltan. Explaining Explainable AI. Explainable AI—"An approximation of... | by Czako Zoltan | The Startup | Medium. URL https://medium.com/swlh/explaining-explainable-ai-b3ca0f8b357b. Last visited 2020-08-09.

A Acronyms

XAI	Explainable artificial intelligence
ML	Machine learning
AI	Artificial intelligence
ANN	Artificial neural networks
FNN	Feedforward Neural Network
DL	Deep learning
DNN	Deep neural network
CNN	Convolutional neural network
TED	Teaching explanations for decisions
MNIST	Modified National Institute of Standards and Technology
SHAP	SHapley Additive exPlanations
LIME	Local Interpretable Model—Agnostic Explanations
AIX360	AI Explainability 360
DBN	Deep Belief Network
LSTM	Long / Short Term Memory Network
GRU	Gated Recurrent Unit
GAN	Generative Adversarial Network
CN	Capsule Network
RNN	Recurrent Neural Network

B Additional Tables

Library	AIX360	Skater	ELI5	LIME	SHAP	InterpretML	XAI	Teller
Python	✓	✓	✓	✓	✓	✓	✓	✓
R				✓	✓			

Library	TreeInterpreter	Alibi	Facets	Keras Visualization Toolkit	iNNvestigate	H2O	DALEX	
Python	✓	✓	✓	✓	✓	✓	✓	
R							✓	

Table B.1: Table that shows compatible languages.

	AIX360	Skater	ELI5	InterpretML	XAI	Teller	Alibi	H2O	DALEX	tf-explain	iNNvestigate
LIME	✓	✓	✓	✓				✓			
SHAP	✓			✓			✓				

Table B.2: Table that shows integration of either LIME or SHAP in libraries.

C User Manual

This chapter describes how to open and run examples. Examples are available as an attachment and on GitHub[1].

AIX360LIME.ipynb File contains example with AIX360 library using LIME (§6.5).

SHAP.ipynb File contains example using library SHAP (§6.4).

Prerequisites

Since we will be running the files in an online service, we will only need internet access.

Usage

To open and run examples, we will be using Google Colab[2].

Load example

After the page loads, the page will show a window with different tabs Figure C.1, we will click the "Upload" tab (1) and then the "Choose File" button (2). After clicking the button, windows explorer shows up and we can locate and open the example.

Run example

To run examples, click on the "Runtime" tab, and click "Run all" to run the whole example.

[1] `https://github.com/fkolenak/XIA_thesis`
[2] `https://colab.research.google.com`

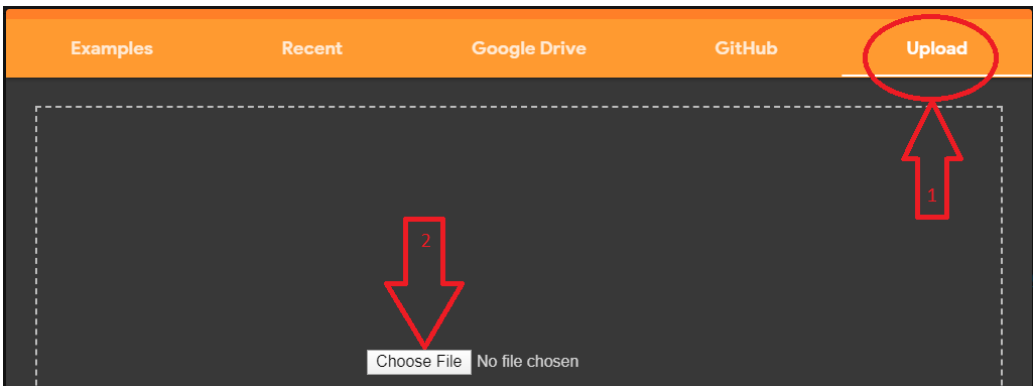


Figure C.1: Open page in Google Colab.

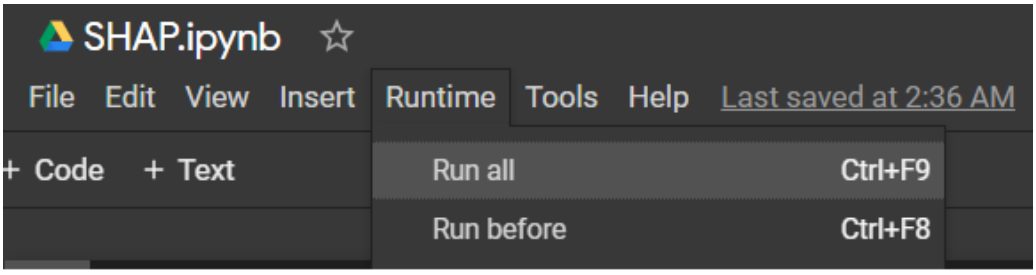


Figure C.2: Control tabs, with selected "Runtime" tab.

D Experiments

D.1 AIX360 library Python compatibility

The most current stable version of AIX360 at the moment (August 2020) is 0.2.0. This version of AIX360 is supported only by the Tensorflow (§2.5) library version 1.14, for compatibility reasons. But there are no specific requirements for Keras (§2.5) library and the latest version of Keras library (version 2.4.3) requires Tensorflow version 2.2.0 and greater. To solve this issue we will need to either override the installation with code Code Listing D.1 or downgrade installed Keras (to version 2.2.4).

The code Code Listing D.1 will install correct version of Tensorflow for Keras and also provide backwards compatibility for Tensorflow. LIME is included in AIX360 package.

```
!pip install aix360
!pip install tensorflow==2.2.0
```

Code Listing D.1: Instalation order of libraries

D.2 AIX360 and LIME experiment versions

Final versions of the libraries after importing are:

- tensorflow 2.2.0
- keras 2.4.3
- aix360 0.2.0
- lime 0.2.0.1

D.3 SHAP experiment versions

Final versions of the libraries after importing are:

- tensorflow 2.3.0
- keras 2.4.0
- shap 0.35.0