Bachelor Thesis

Explaining Artificial Intelligence – An evaluation of the strengths and weaknesses of LIME

Jürgen Dieber

Subject Area: Information Business

Studienkennzahl: 033/561

Supervisor: Dr. Sabrina Kirrane

Date of Submission: 29. June 2020

Department of Information Systems and Operations, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Contents

List of Figures

List of Tables

Abstract

New frameworks in the field of explainable artificial intelligence (xAI) have lately come in the focus of interest, helping to gain insight into the decision making process of complex algorithms, especially in the field of machine learning. In this thesis we evaluate the Local Interpretable Model-Agnostic Explanations (LIME) framework and its performance in making tabular models more interpretable. We apply several state of the art machine learning models on a tabular dataset and compare their performance on a local and global level, using LIME. Additionally, we evaluate the understandability of the LIME output and develop a user experience framework and assess LIME's usability. We found that the tool lacks an updated instruction on its implementation and a clear guideline on how to analyse its output, but overall, the framework's performance proofs satisfying, as it grants interpretability. This theses will facilitate the future application and interpretation of LIME and therefore help to shed more light into black box decisions.

1 Introduction and Research Questions

Since it was first mentioned in 1956 [1], artificial intelligence (AI), and especially its subset machine learning, has steadily made its way into various kinds of industries and aspects of consumers' lives, like healthcare[1][2], transportation[3] and advertisement[4][5]. While machine learning applications are getting more and more advanced, the understanding of how these models work and how decisions are made is becoming less and less. In some applications like order systems or predictive maintenance it may not be necessary to understand the black box decision making, as long as the models' predictions are accurate in the majority of cases. But in circumstances where human lives are involved, like medical diagnosis or self-driving cars, the ability to understand the decision process, is essential.

Explainable AI (xAI) [2] is a generic term used to describe the intention to build and use models that can be interpreted and understood by its users. One approach is to develop powerful and fully explainable models that would make the usage of non-transparent models unnecessary, like deep k-Nearest Neighbours [3] and Teaching Explanations for Decisions [4], another one is to tackle the issue of post-modelling interpretability, to explain the output of well established machine learning models, instead of replacing this models entirely, like SHAP [5], LOCO [6] and MAPLE [7]. In this thesis we focus on the latter approach.

In our literature review in section 2, we identify several post-modeling interpretability frameworks. While many publications introduce new frameworks or variations of existing ones, only little attention is given to the evaluation of their actual performance. The same applies to Local Interpretable Model-Agnostic Explanations (LIME), which is the most frequently mentioned framework in our review. LIME is an open source tool, published by Ribeiro et. al in 2016 [8] and is used as a benchmark in several comparisons [9][10][11]. However, up to now no extensive evaluation has been published about LIME's tabular data function.

[1] https://www.entrepreneur.com/article/341626
[2] https://medicus.ai/de/
[3] https://kodiak.ai/
[4] https://instapage.com/blog/machine-learning-in-advertising
[5] https://www.ezoic.com/

1.1 Research questions

The hypothesis that underlies this thesis is, that explainable AI can enhance the machine learning aided decision process. Based on this we develop the following main question, which is answered by three explicit ones:

How suitable is LIME for the interpretation of tabular models?

To answer this question, we determine the following sub-questions:

- How well does LIME make different models comparable and how can its comparability be improved?
- How interpretable is LIME's output and how can its interpretability be improved?
- How is LIME's usability and how can its usability be improved?

2 Current research on interpretability

In order to gain an understanding of the current state of research in the field of interpretability, we conduct a systematic literature review. We decide to focus exclusively on model agnostic frameworks, which are tools that they are not specialised on one specific machine learning technique, but applicable to several methods. To search for literature, we use the keywords *model agnostic explainability*, *model agnostic interpretability*, *machine learning interpretability*. We find 62 articles about explainable artificial intelligence, of which 37 propose new interpretability methods, evaluate them or extend existing ones. We analyse these publications and aggregate them in Table 1, sorted by the method they are using. We include 21 methods, of which 19 are mathematical methods and three are stand-alone tools applying several methods. We analyse the publications for three additional features: the scope of the interpretability, the type of data the method is tested with and the interpretability evaluation used to assess or compare the method's performance.

Scope In terms of the scope of interpretability, a framework can either be global, meaning it makes different models comparable with each other, by summarizing their performance in specific indicators, or on a local level, giving insight into how a classification in a a single prediction is made. The majority of the reviewed articles focus on the local functionality. Some process locally but can be used for a global comparison and only the activation maximization method [12] and model distillation [13] are exclusively global.

Data Type Our analysis found that while every publication includes some demonstration of the method using a specific data type, which data is being used is very different. Seventeen methods are applied to tabular data, eleven are applied to image data and nine are applied to text data. Only four publications, Koh et al. 2017 [2], Ribeiro at al. 2016 [14] [8] and Sundararajan et al. 2017 [15] include an application of all three data types.

Evaluation Technique Out of the 37 publications, only eleven include an assessment of their tool to measure its performance. We identify two different methods that are used: a *baseline evaluation* and a *user interview*. A *baseline evaluation* is a quantitative evaluation technique, where one or more indicators are measured for each framework which is then used to compare them with each other. For instance, Plumb et al. 2018 [7] uses a self defined causal local explanation metric to compare their framework to LIME. We find ten publications that apply some sort of *baseline evaluation*. The sec-

Method	**Reference**	**Scope**	**Data Type**	**Evaluation Technique**
Activation maximization	[12]	Global	Image	
Counterfactual	[16], [17]	Local	Tabular, Image	
Feature importance	[6] [18]	Global, Local	Image, Tabular	
Fisher kernels	[19]	Local	Image	*Baseline evaluation:* Fisher kernels compared to Influence functions
Frequency map	[11]	Local	Tabular	*Baseline evaluation:* MACEM compared to LIME *User interview:* MACEM compared to LIME
if-then rules	[14], [20]	Global, Local	Image, Tabular, Text	
Influence function	[2]	Local	Image, Tabular, Text	
LIME	[8], [21], [14]	Global, Local	Image, Tabular, Text	
LIME extension	[22], [23], [10], [24], [25], [9]	Local	Image, Tabular, Text	*Baseline evaluation:* SUP-LIME compared to K-LIME; SLIME compared to positive saliency map; DLIME compared to LIME
MAPLE	[7]	Global, Local	Tabular	*Baseline evaluation: MAPLE* compared to LIME
Model distillation	[13]	Global	Tabular	
Parametric statistical tests	[26]	Local	Tabular	
Partial dependence plot	[27]	Global, Local	Tabular	
Prototype and criticism	[28]	Global, Local	Tabular	
Ranking models	[29]	Local	Text	
Relevance scores	[30]	Local	Text	*Baseline evaluation:* LRP compared to TFIDF and uniform
Saliency map	[31], [32] [33], [15], [34], [35]	Local	Tabular, Text, Image	
Sensitive analysis	[36]	Global, Local	Tabular	
Shapley value	[5], [18], [37], [38], [39]	Local	Tabular, Text, Image	*Baseline evaluation:* true shapley value, classical shapley estimations, LIME and ES values *User interview:* SHAP compared to true shapley Value, LIME and shapley sampling
Surrogate models	[8], [40] [41]	Global, Local	Image, Tabular, Text	
Visualisation	[18]	Global, Local	Tabular	

Table 1: Summary of the literature review

ond evaluation technique is a qualitative *user interview*, which for instance in Dhurandhar et al. 2019 [11] ask two professionals to rate a mixed set of interpretability framework outputs given to them. Out of the eleven pub-

lications who evaluate their framework, eight draw a comparison to LIME, from which we can assume, that LIME constitutes a benchmark for interpretability frameworks. However, when it comes to the evaluation of LIME non of the publications about it [8][21][14] use evaluation techniques to assess the frameworks performance and only little attention is given to LIME's tabular interpretation function. Towards this end, this thesis applies LIME on tabular data and evaluates its performance in terms of comparability, interpretability and usability.

3 The interpretation of image classification with LIME

We choose an image classification model[6] to explain the interpretability framework LIME and its application on machine learning models, as using image data offers the advantage of being straight forward and easy to comprehend. This kind of model categorizes an image based on the data it has been trained on. We apply LIME on its output to better understand the process. In the final part we analyse the result LIME presents us with.

The practical work described in this thesis is done in a jupyter environment[7] and runs locally on a MacBook Mid 2014, with one 2,8 GHz Intel Core i5 processor[8], two central process units (CPU)[9] and eight gigabytes of random access memory (RAM)[10].

3.1 What is LIME?

LIME is an open source framework, published by Ribeiro et al. in 2016 [8], aimed to shed light on the decision-making process of any existing machine learning model and therewith establish trust in their user. *Local* means that the framework analyses is observation-specific. It does not give a general explanation of why the model behaves in a certain way, but rather explains how a specific observation is categorised. *Interpretable* stands for being able to understand what a model does. In image classification it shows which part of the picture it considered to make a prediction and when working with tabular data it shows which features influence its decision. *Model-Agnostic* means that it can be applied to any blackbox algorithm we know today or that we might develop in the future. If the model is either a glasbox or not is not taken into consideration as LIME treats every model like a blackbox. *Explanations* are the output that the LIME framework produces. LIME was released with three core functionalities: the image explainer interprets image classification models, the text explainer provides insight into text trained models[11] and the tabular explainer, creates understanding of how features of a tabular dataset are considered for the classification process[12].

[6] https://www.tensorflow.org/tutorials/keras/classification

[7] https://jupyter.org/

[8] https://ark.intel.com/content/www/us/en/ark/products/48496/intel-core-i5-760-processor-8m-cache-2-80-ghz.html

[9] https://www.lifewire.com/what-is-a-cpu-2618150

[10] https://www.digitaltrends.com/computing/what-is-ram/

[11] https://www.tensorflow.org/lite/models/text_classification/overview

[12] https://towardsdatascience.com/pytorch-tabular-binary-classification-a0368da5bb89

3.2 Setup and application of the model

As the basis for our demonstration of LIME, we use a notebook provided by one of the creators of LIME, Marco Ribeiro, on his GitHub account [42]. As a machine learning model Inception_V3[13] a pre-trained model from the python library *keras* [43] is used. Pre-trained means that we can download a ready to use machine learning model and apply it on our data. Inception_V3 is owned by Google Inc.[14] and is a deep neural network that can be used for image classification. It was pre-trained on one-thousand classes[15] of ImageNet[16], an open source labeled image dataset that consists of over fourteen-million images and over twenty-thousand classes. It is an ongoing project in which developers label and add new images and classes to the dataset in an annual competition called ImageNet large scale visual recognition challenge[17].

For the image classification to work, we use the following libraries: `Os` is a package that produces an operating system interface that enables us to directly interact with our operating system. In our case we use it to import the data from our directory. `Keras` is an easy to use API for deep learning[18], which is designed for fast prototyping and research, as it focuses on giving a simple and consistent user experience irrespective of the algorithm you want to use[19]. We import the entire `keras` library as well as packages from it. As our model we use the `keras` version of Inception_V3 a convolutional neural network[20]. `keras.preprocessing`[21] is used to adjust images to make them applicable to our process, the keras `decode_predictions`[22] package for classifying the model and the `matplotlib.pyplot`[23] package is needed every time we process our model. The `numpy`[24] library is an essential package for many kinds of computational work with python which we use to apply our classification model.

We first call the `inceptionV3` function on the *inception_v3* package and set it to `inet_model`. Then we define the `transform_img_fn` Listing 1 function which serves the following purposes: First, it processes the image to

[13]https://keras.io/applications/#inceptionv3
[14]https://www.fast.ai/2017/01/03/keras/
[15]https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
[16]http://image-net.org/about-overview
[17]http://image-net.org/challenges/LSVRC/2016/index
[18]https://keras.io/
[19]https://keras.io/why-use-keras/
[20]https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
[21]https://keras.io/preprocessing/image/
[22]decode_predictions vgg16
[23]https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.html
[24]https://numpy.org

category number	category name	probability
n02859443	boathouse	0.9577018
n09332890	lakeside	0.0033688135
n04238763	slide_rule	0.0007693328
n07715103	cauliflower	0.00064561074
n04590129	window_shade	0.00050171226

Table 2: Classification output of the first image

a form applicable to LIME which is 299 X 299 pixel and assigns it to the variable `img` (line 5), then it transforms the image into an array, which is furthermore preprocessed by the `inc_net` (line 8). `Transform_img_fn` returns the output, stacked in vertical sequences, processed by the numpy `vstack` function (line 10).

Listing 1: The image transformer

```
1  # Processes the picture to make it applicable
2  def transform_img_fn(path_list):
3      out = []
4      for img_path in path_list:
5          img = image.load_img(img_path, target_size=(299, 299))
6          x = image.img_to_array(img)
7          x = np.expand_dims(x, axis=0)
8          x = inc_net.preprocess_input(x)
9          out.append(x)
10     return np.vstack(out)
```

Application of the machine learning model We use our model to classify buildings. The ImageNet dataset contains the classes *boathouse*, *church*, *barn* and *lakeside* amongst others, therefore our model can be tested to identify them. We examine its accuracy with two pictures both of them displaying a boathouse and water. Based on the Cambridge dictionary definition of a boathouse, it is *"a small building at the side of a river or lake, in which boats are kept"* [44]. The examples are chosen based on this definition.

We load Figure 1 [45], a picture of the Stafford's boatshouse into our model using `os.path.join` which prints the output of the five highest rated categories displayed in Table 2. The category *boathouse* is rated the most likely class with 95.78% and the next possible category *lakeside* has a weight of 0.34% which is therefore not considered likely anymore.

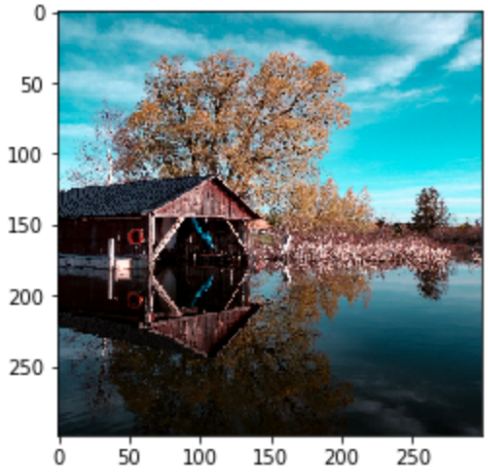

Figure 1: The first image example

3.3 Applying LIME on the classification output

Now we apply LIME on the machine learning output to understand the model's prediction. We define the `explainer` by loading and running the `LimeImageExplainer` function, displayed in the first line of Listing 2. For it to work, we change the parameter setting `top_labels` (line 5), as the default value results in an error from 5 to 1000 labels, which is the number of all classes the model contains. The `%%time` command (line 7) comes with python and shows the processing time of a function, which in our case is ten minutes and thirty-eight seconds.

Listing 2: The LIME image explainer

```
1 explainer = lime_image.LimeImageExplainer()
2
3 # Hide color is the color for a superpixel turned OFF.
    Alternatively, if it is NONE, the superpixel will be replaced by
    the average of its pixels
4
5 explanation = explainer.explain_instance(images[0],
    inet_model.predict, top_labels=1000, hide_color=0,
    num_samples=1000)
6
7 %%time #shows the processing time the function takes
```

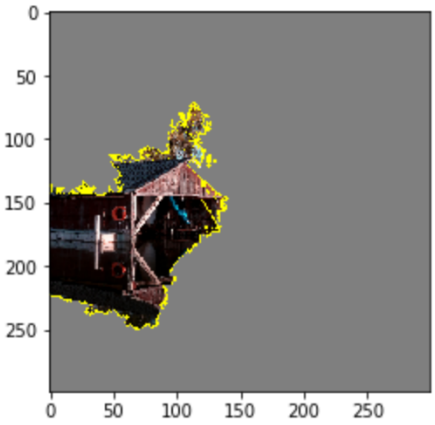

Figure 2: The image processed by the model

In Figure 2 we display the first interpretation of LIME by applying the `explanation` to produce the five most important parts of the image the model bases its top classification on, in our case the *boathouse*. We can see that the model considers exclusively the left side of the picture where the building is, branches of the tree behind, as well as the buildings reflection in the water, which tells us that a house and the nature around it are the main indicators used in order to classify this image as a boathouse.

To display the whole picture, with the most important pixels coloured, we set the parameter `hide_rest=FALSE` and `positive_only=TRUE`. The effect of the change in settings can be inspected in Figure 3. The fields that contributed positively to the chosen label are displayed in green and those influencing negatively toward the top prediction are displayed in red. As our prediction has a weight of 95.78% no red spot can be identified.

In Figure 4 we display only features that have a weight of 25% or more by including the parameter `min_weight=0.25` in the function. In our case the center of the building and its reflection are considered the strongest. Therefore we can conclude that the model is primarily influenced by this fraction of the picture. This specification helps us to identify more clearly what factor the model is focused on. We then increase the `min_weight` gradually until at 34% no superpixel is shown anymore.

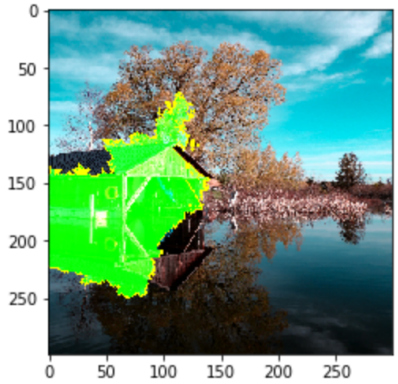

Figure 3: The top five superpixel

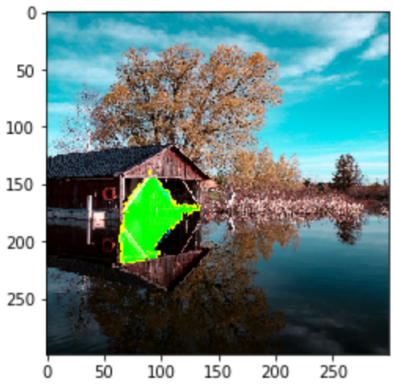

Figure 4: Pixel with ≥25% weight

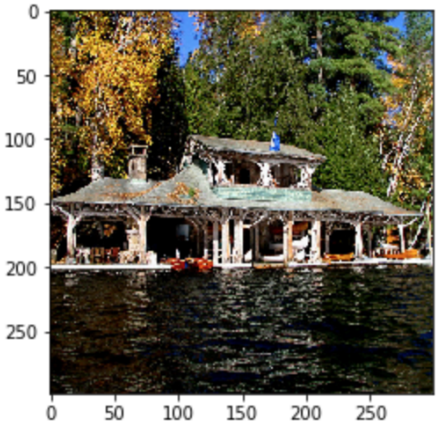

Figure 5: The second image example

In order to draw a comparison we analyse a second example of a boathouse, borrowed from Wikipedia's definition of a boathouse [46]. Figure 5 displays a house and a waterside, not unlike our first example. We apply our model on it and get the results shown in Table 3: The category *boathouse* is rated the most likely class, with 62.86%. The next possible category *lakeside* has a likelihood of 21.61%, which is still considerably high, but the third category, *fountain*, only reaches 0.34% and is therefore not considered likely anymore.

category number	category name	probability
n02859443	boathouse	0.62867475
n09332890	lakeside	0.21610458
n03388043	fountain	0.0033783945
n03874293	paddlewheel	0.0027901048
n02099712	labrador_retriever	0.00029043932

Table 3: Classification output of the second image

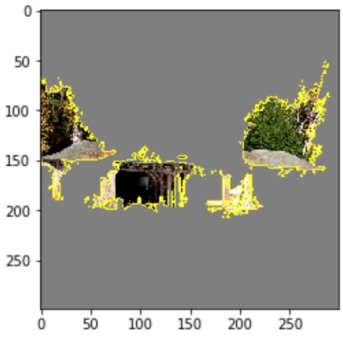

Figure 6: The top five superpixel

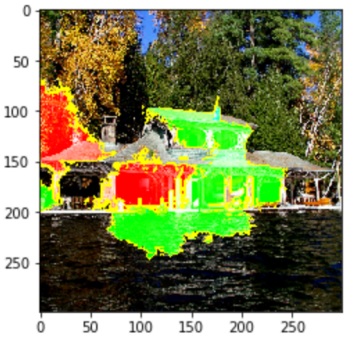

Figure 7: The second prediction

Compared to the first image, we now have two options that are considered possible. To understand why the model regards this image differently we make use of the `LIMEImageExplainer`. We look at which factors the model regards differently and what led to a change from an unequivocal top category in the first image to two highly rated categories in the second one. We again apply the `explanation` on the image which results in Figure 6, to see what differs from Figure 1, where the mode made a definite classification. This time the model mainly considered the center of the picture, parts of the roof and trees in the background, as well as the storage for a boat, something very characteristic for a boathouse[25].

Finally, we take a look at the analysis of the second highest rated prediction, *lakeside*, with 21.61%. By including `106` in the `explanation` function and changing `positive_only` from `true` to `false` we can view in Figure 7 the most positively influencing pixel for the second highest weighting category, *lakeside*, in green and the most negatively influencing pixel in red. We

[25]https://en.wikipedia.org/wiki/Boathouse#cite_note-Evesham-1

can see that not only the parts of the picture capturing the lake positively contribute to this category, but also parts of the boathouse. This indicates, that the image data the model has been trained with to classify a *lakeside* displayed similar buildings, making it one of the primary characteristics of this picture.

3.4 An evaluation of the LIME image explainer

LIME's performance in interpreting images is very impressive as it makes the model's decision making process transparent for the user. LIME's method to generate superpixel, aggregated areas that have been judged alike by the model and make them visible to the user, is an easy to comprehend solution. Even in a case like Figure 5 which offers two potential answers, it illustrates the factors considered clearly, enabling us to understand why one category was chosen over the other. The available options to display only the relevant pixel, colour them in red and green or only show superpixels of a certain weight, help to get a more detailed insight into the classification process. All in all, the `LIMEImageexplainer` helps to identify possible bias and thereby establishes trust between the classification model and those that need to interpret the result.

4 The classification of tabular data

In this chapter we form the basis to evaluate the *LIMETabularExplainer*. We apply four classification models, the decision tree, the logistic regression, the random forest and the extreme gradient boosting on tabular data and analyse the output using the confusion matrix, accuracy score and receiver operating characteristic curves.

4.1 Tabular data retrieval and pre-processing

For our tabular data analysis we use the *Rain in Australia* data-set from Kaggle[26]. Before the algorithm is trained, we work through the different variables step by step to fully understand their meaning. Then the dependencies amongst each other are evaluated and those that could potentially bias the models or have too many missing values are removed.

4.1.1 The setup

The implementation process is split into data processing, the setup of the algorithm, the actual implementation and the application of LIME.

Data processing To load and process the data we make use of the pandas[27] library. It is one of the most frequently used python libraries in the field of data science, as it allows to import data into very well structured data frames and adapt it to your needs. The numpy[28] package is also a very popular python library, used for linear algebra and multidimensional containers. In this thesis it is used to set up the confusion matrix.

Setup of the machine learning algorithm Before we can train our model, we need the following: The `OneHotEncoder`[29] function ensures that the variable can be processed by the algorithm, by adding a dummy variable to the data-frame for every category the variable contains. The `ColumnTransformer`[30] function allows us to process a single column or a column subset independently.

[26] https://www.kaggle.com/jsphyg/weather-dataset-rattle-package
[27] https://pandas.pydata.org/pandas-docs/stable/
[28] https://numpy.org/doc/
[29] https://bit.ly/2RnWZuK
[30] https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html

To set up our models, the following packages are used: The `OneHotEncoder`[31] function makes a categorical variable process-able for the algorithm by adding a dummy variable to the data-frame for every category the variable contains. The `ColumnTransformer`[32] function allows us to process a single column or column subset independently. For establishing the model we use the `Pipeline`[33] package which simplifies our code by aggregating several steps and performing them in a sequence. The `GridSearchCV`[34] function which combines a grid search[35] with an estimator[36], is applied to tune our hyper-parameters. It allows us to set a range instead of a specific value for a parameter and automatically picks the best performing one of all training rounds. The scikit-learn `train_test_split` package[37], is used to separate our data-set into training and testing data.

The models To compare varying algorithms we apply the following methods: decision tree[38], logistic regression[39], random forest[40] and XGBoost[41] which are all part of the sklearn library[42].

Interpretation To compare the performance of our models we use the state of the art methods receiver operating characteristic curve[43], classification report[44] and the accuracy package[45]. In the end, the `LimeTabularExplainer`[46] is used as an interpretability framework.

4.1.2 Description of the dataset

The Australia rain prediction dataset contains twenty-four features of which seven are categorical, including the binary target variable and seventeen are

[31] https://bit.ly/2RnWZuK
[32] https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html
[33] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
[34] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[35] https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998
[36] https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
[37] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[38] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[39] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[40] https://bit.ly/2RnWZuK
[41] https://xgboost.readthedocs.io/en/latest/
[42] https://scikit-learn.org/
[43] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
[44] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[45] https://bit.ly/2xtZ2XP
[46] https://lime-ml.readthedocs.io/en/latest/lime.html

variable name	sample input	type	non-null-values
Date	2008-12-03	categorical	142193
Location	Albury	categorical	142193
MinTemp*	13.4	numerical	141556
MaxTemp*	25.1	numerical	141871
Rainfall*	0.00	numerical	140787
Evaporation	23	numerical	81350
Sunshine	11	numerical	74377
WindGustDir*	W	categorical	132863
WindGustSpeed*	44.0	numerical	132923
WindDir9am*	NW	categorical	132180
WindDir3pm*	W	categorical	138415
WindSpeed9am*	25.0	numerical	140845
WindSpeed3pm*	8.0	numerical	139563
Humidity9am*	25.0	numerical	140419
Humidity3pm*	22.0	numerical	138583
Pressure9am*	1007.7	numerical	128179
Pressure3pm*	1007.1	numerical	128212
Cloud9am	2.0	numerical	88536
Cloud3pm	8.0	numerical	85099
Temp9am*	16.9	numerical	141289
Temp3pm*	21.8	numerical	139467
RainToday*	Yes	categorical	140787
RISK_MM	0.2	numerical	142193
RainTomorrow*	No	categorical	142193

Table 4: An overview of the datasets' features

numerical. In total the file consists of 142,193 rows, taking about twenty-six megabyte of disk-space. In Table 4 a summary of the dataset is given.

Categorical variables The categorical features, which are all in string format, consist of *Date*, *Location*, *WindGustDir*, *WindDir9am*, *WindDir3pm*, *RainToday* and the dependent variable *RainTomorrow*. The variable *Date* is structured in the YYYY-MM-DD format and contains a timeframe between 1st of November 2007 to 25th of June 2017. *Location* includes the city the observation has been recorded in, including fourty-nine cities in Australia, but no coordinates or further areal separation. *WindGustDir* describes the direction of the strongest wind gust, a wind stream that lasts several seconds and exceeds a speed of twent-nine kilometers per hour. *WindDir9am* and

WindDir3pm both represent a cardinal point at the exact time. *RainToday* is a boolean that is 1 if the rainfall (in mm) in the twenty-four hours before 9am exceeds 1mm. Otherwise it is set to 0. The target variable *RainTomorrow* is also a boolean, either 0 for it is going to rain tomorrow or 1 if it is not going to rain tomorrow.

Numerical variables *MinTemp* and *MaxTemp* represent the minimum and maximum temperature degree at a given day and *Temp9am* as well as *Temp3pm* the temperature of the given moment. Like all numerical variables, they are in float format. *Rainfall* shows the amount of rain in millimeters that has been measured. For *Evaporation* the class A pan evaporation[47] is measured, which is a method for measuring how much water vaporises on a given day, strongly influenced by the amount of sunlight, temperature and windspeed. *WindGustSpeed* captures the speed the strongest recorded gust has reached and both *WindSpeed9am* and *WindSpeed3pm* records the actual speed at the given point of time, respectively. *RISK_MM* is a calculation of the expected rainfall of tomorrow, calculated by today's values. It is used to create the target variable *RainTomorrow*. *Cloud9am* and *Cloud3pm* are measuring the fraction of sky obscured by clouds at the given time, respectively. It is measured in oktas[48], which are a unit of eigths. It records how many eights of the sky are obscured by clouds. A 0 measure indicates completely clear sky whilst an 8 indicates that it is completely covered. The only variables measured in percentage are *Humidity9am* and *Humidity3pm*. *Pressure9am* and *Pressure3pm* each show the atmospheric pressure, measured in hectopascal pressure unit[49], reduced to the mean sea level.

The column non-null-values in Table 4 includes information from the `info` function[50] of the pandas library. It returns the format of the feature as well as the number of non-null values , which vary from feature to feature. While some variables have $<5\%$ missing values (e.g. *MinTemp*, *MaxTemp*, *WindSpeed3pm*), others miss over 40% (e.g. *Cloud9am*, *Sunshine*).

[47] https://theconstructor.org/water-resources/evaporation-and-its-measurement/4575/
[48] https://www.metoffice.gov.uk/weather/guides/observations/how-we-measure-cloud
[49] https://www.sensorsone.com/hpa-hectopascal-pressure-unit/
[50] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html

4.1.3 Processing the directions

In order to be able to work with the direction giving variables *WindGustDir*, *WindDir9am* and *WindDir3pm* we have to convert the string to a float. Prior to including the directional variables in our input data *WindGustDir*, *WindDir9am* and *WindDir3pm*, the cardinal points have to be converted to processable numbers. We therefore assign a number between 1 and 16 to each of the possible directions, for example the direction North is represented in the dataset by the letter N and is changed to 1. North-North-East was represented by NNE and is replaced by the number 4. We replace the values of all three features by using the pandas `map` function [51].

Converting binary variables from string to float In order to process the binary variables *RainToday* and *RainTomorrow*, we convert them from Yes/No to 1/0, using the pandas `map` function.

Dropping variables and missing values Considering the variables meaning, we decide to exclude two from our analysis. *Risk_MM* shows how much it will rain on the next day, making it highly predictive for our dependent variable. The variable *Date* was excluded as a date has no influence on the natural phenomenon of rain and does not have any effect on it, as it is a generic measurement of our society to structure a time-span. Furthermore *Evaporation*, *Cloud3pm*, *Cloud9pm* and *Sunshine* are dropped because they have 40-50% missing values. To remove all of them, we use the pandas `drop`[52] function. Furthermore, we have to get rid of missing values within the features we decide to keep, for which we use the pandas `dropna`[53] function.

After preprocessing In the end our data frame consists of seventeen features, twelve numerical and five categorical, including the target variable, all in a float format, which are marked with a * in Table 4. This results in a total of 112.925 observations to train and test our model with.

4.2 Setup of the machine learning models

In order to make our model trainable, we need to preprocess our features further. We use the `ColumnTransformer` function[54] to process the features. By using `passthrough` on the numerical variables it will leave them untouched.

[51] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.map.html
[52] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html
[53] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
[54] https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html

The categorical features instead are processed by the `One-Hot-Encoder` which converts categorical variables into a form that can be used by a statistical model. Instead of using a variable with several numbers representing categories, each variable is split into one variable per possible outcome, each of them either being 1 if the category is true or 0 if this category is false.

We further build a `pipeline` object[55], displayed in Listing 3, to apply the preprocessor on the data and sequentially build our model based on its structure. This enables us to perform a sequence of different transformations and to give each algorithm a customised setting while being able to cross-validate each setting-combination during the training process. We use the pipeline for different functions: `Class_weight="balanced"` is applied to the decision tree (line 4), the logistic regression (line 8) and the random forest (line 12), to correct the unbalanced dataset and to train the the models with an even amount of target variable outcomes. For the XGBoost we apply `scale_pos_weight=(1-y.mean())` which has the same effect. `Solver="liblinear"`, which we apply on the logistic regression (line 8), is the recommended setting when working with binary classification problems as it applies automatic parameter selection[56] to the model. The two parameters `random_state` and `n_estimator` guaranty consistency for the logistic regression and the random forest, respectively. After setting `random_state=42` (line 8) the logistic regression always uses the same state and when `n_estimator=100` (line 12) is applied, the random forest produces a constant number of trees, instead of constantly changing. The parameter `n_jobs` does not influence the computation itself, but `n_jobs=-1` (line 12 and 17) tells the CPU to use 100% of all available cores, reducing the processing time.

Listing 3: The model pipeline

```
1 # Decision Tree
2 dt_model = Pipeline([("preprocessor", preprocessor),
3                ("model",
                    DecisionTreeClassifier(class_weight="balanced"))])
4
5 # Logistic Regression
6 lr_model = Pipeline([("preprocessor", preprocessor),
7                ("model", LogisticRegression(class_weight="balanced",
                    solver="liblinear", random_state=42))])
8
```

[55] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[56] https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-defintions

```
# Random Forest
rf_model = Pipeline([("preprocessor", preprocessor),
                ("model",
                    RandomForestClassifier(class_weight="balanced",
                    n_estimators=100, n_jobs=-1))])

# XGBoost
xgb_model = Pipeline([("preprocessor", preprocessor),
                      # Add a scale_pos_weight to make it balanced
                     ("model", XGBClassifier(scale_pos_weight=(1 -
                        y.mean()), n_jobs=-1))])
```

The `train_test_split`[57] function is used to break our data into different parts, namely training and testing data. The training data consists of the variable *x_train*, which contains 70% of the instances, excluding the target variable and *y_train* only contains the target variable of the same instances as *x_train*. The test data is separated into *x_test* and *y_test*, with the former containing the other 30% of the data, including all variables except the target variable, which again is extracted to the latter. For training our model with the same target variable distribution as the dataset, we include `stratify=y` in the `train_test_split` function, resulting in the same distribution of y in the training and test data, respectively.

4.3 The application of machine learning models

In our experimental setting, we test several algorithms and interpret the differences in the results. We start with a decision tree, based on the assessment of Molnar [47] and Hara et al. [48] as it offers a simple and easy to interpret solution, where you can go through each taken decision step by step and therefore not necessarily need an explainable AI framework. We then pick three state of the art solutions for classification tasks, logistic regression, random forest and XGBoost which are all based on different mathematical techniques: stacking, bagging and boosting[58]. This chapter gives more insight into how the algorithms are applied and what steps for the analysis are taken.

[57] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[58] https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Listing 4: The decision tree model

```
1 \lstset{floatplacement=tp}
2 #Optimizing the parameters by using cross validation
3 gs = GridSearchCV(dt_model, {"model__max_depth": [3, 5, 7],
4                              "model__min_samples_split": [2, 5]},
5                   n_jobs=-1, scoring="accuracy")
6
7 gs.fit(X_train, y_train)
```

4.3.1 The decision tree algorithm

A decision tree is a supervised learning method [49], which is used to predict an outcome by answering a set of questions and therefore coming to a decision, based on the features it received as an input. These answers can be manually traced from beginning to end and are therefore very interpretable by nature. This high interpretability makes it a good benchmark for comparison with blackbox models. Although we examine the decision tree in detail, the other algorithms are analogous.

We train the decision tree using the `GridSearchCV` function, operating it mostly on default settings, the code being displayed in Listing 4. The aspects we alter are the `model_max_depth` (line 2), which represents how deep a tree is allowed to grow. The deeper a tree becomes the more splits it has and the more information it can capture. But it is also more likely to adapt specifically to the training_data, losing the ability to predict other datasets, a statistical phenomenon called overfitting[59]. We set it to `[3, 5, 7]` splits so the grid search can single out the best one to use. `Model_min_sample_split` (line 3) determines the minimum sample size required to split an internal note, which is per default set to a sample size of `2` and we widen the spectrum by adding `5`. `N_jobs=-1` (line 4) tells the CPU to use 100% of all available cores, reducing the processing time. The parameter `scoring="accuracy"` indicates, that the model's goal is to maximise its accuracy. Lastly the `fit` function (line 6) is used to train the model defined in the `GridSearchCV`.

The classification report The sklearn classification report, displayed in Table 5, provides us with further insight into the performance of the model. The *precision* describes how often the model was correct in classifying an observation as positive. It is the result of the true positives, divided by the

[59] https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

target/metric	precision	recall	f1-score	support
not going to rain	0.91	0.81	0.86	26372
going to rain	0.53	0.73	0.61	7506
accuracy			0.79	33878
macro avg	0.72	0.77	0.74	33878
weighted avg	0.83	0.79	0.80	33878

Table 5: The decision tree classification report

sum of true positives and false positives, adding up to 91% for the outcome that it is not going to rain and 53% for that it is going to rain. For the *recall* measurement, the performance of the variables is more similar. It consists of the true positives divided by the sum of true positives and false negatives, 81% and 73%, respectively. The *f1-score* tells us what percentage of positive prediction is correct, including the recall and *precision* into its measurement. The *f1-score* consists of two times the *precision* * *recall* divided by the sum of *precision* and *recall*. The decision tree delivers a *f1-score* of 86% for that it is not going to rain and 61% for it is going to rain. The column *support* is a count for the instances of the target variable, in the upper part of the matrix for the outcome not going to rain and going to rain, in the lower part it shows the total of 33878 observations that have been used for the test.

The bottom block of the `classification report` shows the *accuracy score* and the *macro* and the *weighted average* of the three indicators above plus the *f1-score*. The *macro score* represents the overall performance of the indicator, meaning the average. The *macro precision* reaches 82%, the *macro recall* 71% and the *macro f1-score* 74%. The *weighted average* is, as the name indicates, the respective score times its number of instances, for example, the 0.85% *weighted average precision* result from the target variable not going to rain, having a score of 91%, with 26372 observations and and 53% of target variable going to rain with 7506 instances. Overall, the model scores highly in most indicators, but seems to struggle with correctly predicting, as a precision of 53% and an f-1 score of 61% leave room for improvement. A more detailed evaluation of classification report indicators, including a comparison of all four models, is described in section 5.

The receiver operating characteristic curve Another state of the art tool to measure the validity of classification results is the receiver operating characteristic curves (ROC) [50]. As displayed in Figure 8, it is a graph

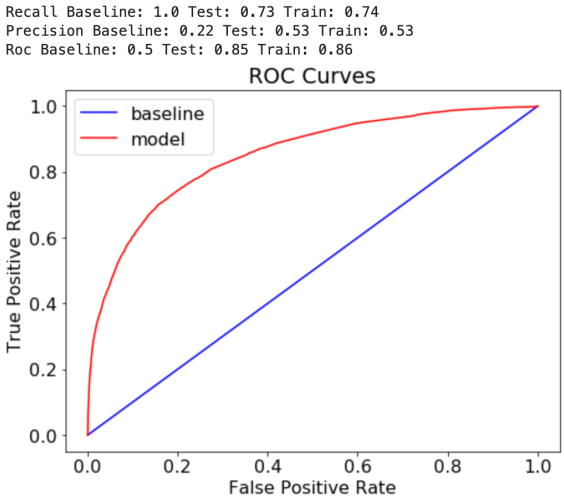

Figure 8: The decision tree ROC curve

showing two curves, the upper one is the ROC curve, posing a probability, the lower one is the *baseline*, which separates the ROC and the area under the curve (AUC), which is a measurement for separability. Similar to the `classification report`, the ROC curve uses *precision* and *recall* for its measurement but due to its graphical display curves of different models can be compared visually to each other. The further to the upper left corner the curve bends, the better the classification. The AUC measures the general accuracy, meaning how well a model can differentiate between classes. For the AUC the following rule holds true: the closer its value is to 1, the better the model is able to correctly classify. If the value is 0.5 it means that the model is not better than randomly guessing and a value of close to 0 means that the model is doing the classification upside down [60]. In the case of our decision tree, the *baseline* performs with 0.85 on our test-data and the model can therefore be interpreted as reliable.

4.3.2 The random forest algorithm

A random forest is an aggregation of various decision trees, calculating decisions over and over again until a certain amount of sequences is calculated

[60] https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ https://www.jstor.org/stable/2531595?seq=1

Listing 5: The random forest model

```
\lstset{floatplacement=tp}
gs = GridSearchCV(rf_model, {"model__max_depth": [10, 15],
                             "model__min_samples_split": [5, 10]},
                  n_jobs=-1, scoring="accuracy")

gs.fit(X_train, y_train)
```

[51]. Then the algorithm utilises the decision made by the majority which are then used to build a model. The assumption is that each decision tree might have its individual miscalculations, but a large number of trees will annul each others errors. Therefore, a random forest is the statistical equivalent to the wisdom of the crowd [52]. For our model, displayed in Listing 5, we set the same parameters as those of the decision tree, except for increasing the numbers of `max_depth` (line 1) to `[10, 15]` as the default allows them to deepen until all leaves are pure, which might lead to overfitting[61] and the `minimal_sample_split` to `[5, 10]` as we want to give the `GridSearchCV` a range of values to choose from. All other parameters stay unchanged.

The results In terms of accuracy the random forest reaches 79.97%. The classification report, as displayed in Table 6, shows that the *precision* between the two outcomes of *RainTomorrow* diverge quite heavily, with 92% and 53% for it is not going to rain and it is going to rain, respectively. The *recall* of 82% and 74% is much closer and the *f1-score* again diverges, 86% and 62% for it is not going to rain and it is going to rain, respectively. In terms of *macro average* the *precision*, *recall* and *f1-score* reach 73%, 78% and 74%, while the *weighted average* scores 83%, 80% and 81%, respectively.

The random forest ROC delivers a baseline performance of 86% on the test data as can be seen in Figure 9.

[61] https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

target/metric	precision	recall	f1-score	support
not going to rain	0.92	0.81	0.86	26372
going to rain	0.53	0.75	0.62	7506
accuracy			0.80	33878
macro avg	0.72	0.78	0.74	33878
weighted avg	0.83	0.80	0.81	33878

Table 6: The random forest classification report

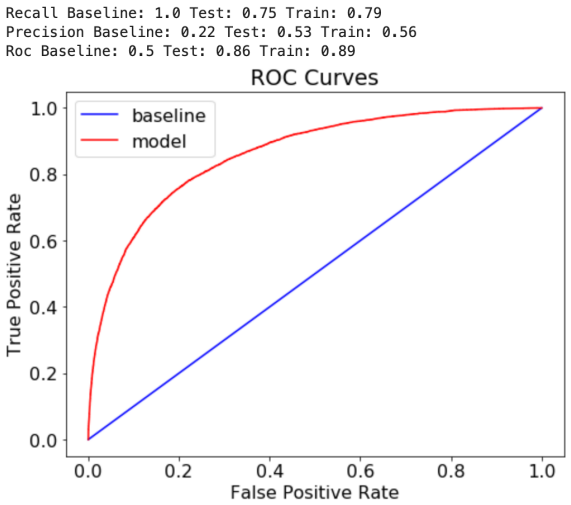

Figure 9: The random forest ROC curve

4.3.3 The logistic regression algorithm

The second model we use for comparison is the logistic regression. Due to it being a regression, it is sometimes referred to as a glasbox model [47], like the decision tree, and therefore easier to interpret than more complex models like the XGBoost. To build the model we use the `GridSearchCV` function and specify only certain parameters (see Listing 6). The first one, `model__C`, is a regularizing parameter is used to avoid overfitting. As the default value is 1.0 and we want to give the `GridSearchCV` a range to experiment with, we add 1.3 and 1.5. `N_jobs= -1`, `cv` and `scoring` are the same as in the decision tree.

Listing 6: The logistic regression model

```
\lstset{floatplacement=tp}
gs = GridSearchCV(lr_model, {"model__C": [1, 1.3, 1.5]}, n_jobs=-1,
    scoring="accuracy")
gs.fit(X_train, y_train)
```

target/metric	precision	recall	f1-score	support
not going to rain	0.92	0.80	0.86	26372
going to rain	0.52	0.77	0.62	7506
accuracy			0.79	33878
macro avg	0.72	0.79	0.74	33878
weighted avg	0.84	0.79	0.80	33878

Table 7: The logistic regression classification report

The results In terms of performance, the logistic regression delivers an *accuracy* of 79%. The *precision* of the two depended variables, displayed in the Table 7, appears to be mixed, 92% indicating it is not going to rain and 52% indicating it is going to rain. The *recall* results are almost identical with 80% and 77% and the *f-1 scores* are somewhere in the middle, with 86% and 62%, indicating it is not going to rain and it is going to rain, respectively. The *macro average* returns 72% *precision*, 79% *recall* and 74% *f1-score* and the *weighted average* returns 84%, 79% and 80% respectively.

The ROC curve, in Figure 10, reaches a value of 87% in its test run, which is a positive indicator [62].

4.3.4 The extreme gradient boosting algorithm

The extreme gradient boosting algorithm (XGBoost) was developed by Tainqi Chen [53] and belongs to the distributed machine learning community. The algorithm is focused on computational speed and model performance which are key factors when considering which model to use in the field of applied machine learning. The XGBoost uses gradient boosted decision trees for its calculations. Boosting is a technique where new models are added to correct the errors made by previous models, based on the assumption, that a set of models performs better than a single model. Models are added sequentially

[62] https://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/

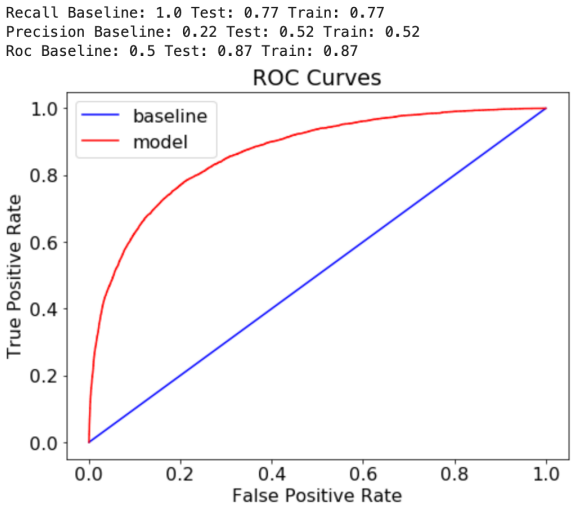

Figure 10: The logistic regression ROC curve

until no further improvements can be made. The XGBoost has gained lots of recognition over the last years as it has been used for a series of wins at international data science competitions like Kaggle[63] [64].

As displayed in Listing 7, we set the default of `model_max_depth` to `[5, 10]`, with the intention to keep the maximum depth of trees in a range which neither tends to overfit nor needs too much processing power[65]. `Min_child_weight` is altered to control possible overfitting, as a high number prevents a model from learning relations which only exist in the training data, however, if the number is too high it can lead to underfitting, therefore we set it to `[5, 10]` as well[66]. `Model__n_estimators` represents the number of gradient boosted trees we use, which results in the number of rounds we process before choosing the best one. As this is one of the main influences in terms of computing power, we pick `[25]`, which is a quarter of the default

[63] https://www.quora.com/What-machine-learning-approaches-have-won-most-Kaggle-competitions

[64] https://medium.com/syncedreview/tree-boosting-with-xgboost-why-does-xgboost-win-every-machine-learning-competition-ca8034c0b283

[65] https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/

[66] https://medium.com/data-design/xgboost-hi-im-gamma-what-can-i-do-for-you-and-the-tuning-of-regularization-a42ea17e6ab6

Listing 7: The XGBoost model

```
\lstset{floatplacement=tp}
gs = GridSearchCV(xgb_model, {"model__max_depth": [5, 10],
                              "model__min_child_weight": [5, 10],
                              "model__n_estimators": [25]},
                  n_jobs=-1, scoring="accuracy")

gs.fit(X_train, y_train)
```

target/metric	precision	recall	f1-score	support
not going to rain	0.86	0.96	0.91	26372
going to rain	0.79	0.46	0.58	7506
accuracy			0.85	33878
macro avg	0.82	0.71	0.74	33878
weighted avg	0.85	0.85	0.84	33878

Table 8: The XGBoost classification report

value [100], as we train the model on a local machine[67]. All other factors stay equal to the settings of the previous models.

Â´

The results The results from the classification report, displayed in Table 8 are as followed: The *precision* reaches 86% and 79% percent for the dependent variable 0 and 1, *recall* 96% and 46% and *f1-score* 91% and 58%. The *macro average* resulted in 82%, 71% and 84% and the *weighted average* in 85%, 85% and 84%, respectively. The model achieves an overall *accuracy* of 85% on the testing data.

The ROC curve in Figure 11 runs the smoothest compared to the ROC of the other models, with a result of 88%.

[67] https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/

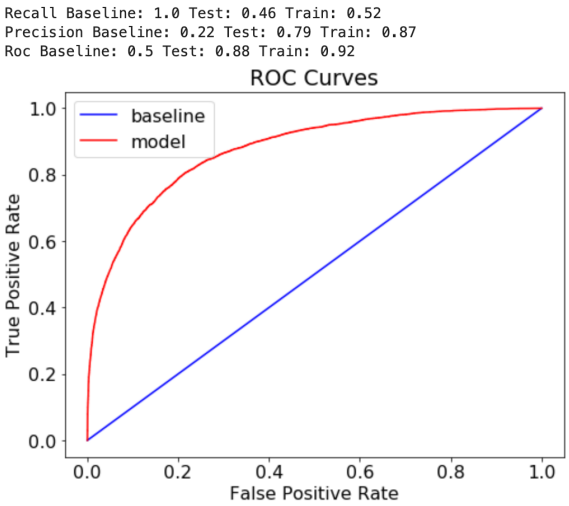

Figure 11: The XGBoost ROC curve

5 Interpretation of the output and the application of LIME

This chapter focuses on explaining the output presented in the previous sections and consists of three parts: in the first part we analyse the machine learning results and compare the models' performance, using the accuracy score, the classification report and the ROC curve; In the second part we apply LIME on the tabular models and describe the output, while in the third part we conduct a quantitative analysis of fifty aggregated LIME observations to assess the models' performance with LIME on a global level.

5.1 Analysis of the machine learning output

The logistic regression as well as the decision tree perform slightly worse than the random forest in all performance indicators. Table 9 is an aggregation of all conventional performance measures we use to evaluate our models. The scores either refer to the target variable it is not going to rain (0) or it is going to rain (1) as well as the weighted scores (w) and in terms of the ROC the training baseline value (tr).

Overall it is notable that the performances of the decision tree, random forest and logistic regression are very similar while the XGBoost performance differs significantly.

In this comparison, the XGBoost delivers the highest values with a 85% *accuracy*, *weighted average scores* of 85% *precision*, 85% *recall* as well as 84% *f1-score*. But it's weak performance in classifying that it is going to rain correctly, can be seen in a low *recall (1)* and *f1-score (1)* score with 46% and 58%, respectively. It is worth noting that the high difference in the *recall* scores for the respective target variable might be caused by unbalanced testing data, which is something we would like to further explore in future work.

The logistic regression offers the highest *recall (1)*, in 77% of the positive observations it predicts correctly that it is going to rain, with a weighted *recall* of 79%. In terms of *f1-score (1)* the logistic regression and the random forest score equally 62% which is four percentage points higher than the XGBoost with 58%.

Furthermore, comparing the ROC curves shows a similar performance of the models, with the XGBoost scoring an 88% *ROC baseline*, the logistic regression 87%, the random forest 86% and the decision tree 85%.

To summarize, the decision tree performs worst in all metrics. The random forest and the logistic regression never differ more than two percentage

	accuracy	precision (0)	precision (1)	recall (0)	recall (1)	f1-score (0)	f1-score (1)	precision (w)	recall (w)	f1-score (w)	ROC-baseline (tr)
decision tree	0.79	0.91	0.53	0.81	0.73	0.86	0.61	0.83	0.79	0.80	0.85
random forest	0.80	0.92	0.53	0.81	0.75	0.86	0.62	0.83	0.80	0.81	0.86
logistic reg.	0.79	0.92	0.52	0.80	0.77	0.86	0.62	0.84	0.79	0.80	0.87
XGBoost	0.85	0.86	0.79	0.96	0.46	0.91	0.58	0.85	0.85	0.84	0.88

Table 9: A model comparison using conventional methods

points in any of the metrics and are therefore performing very similar. Only the XGBoost outperforms the others in several metrics, but scores significantly lower when it comes to predicting the outcome of a positive observation. To answer the question which model should be deployed therefore requires the knowledge of which trade-off is preferred: a higher accuracy and more accurate prediction of true negatives would stand in favor of the XGBoost and the need for a more accurate prediction of true positives would favor the random forest or the logistic regression.

5.2 The application and evaluation of the LIME Tabular Explainer

After using conventional methods to analyse the models' performance, we now apply and interpret LIME to gain further insight into the decision making process.

Set up of the explainer The main function LIME offers is called `explainer` which enables us to call a specific observation and get an interpretation as a result. Prior to being able to explain an observation, we need to convert the output into a certain format, which is displayed in Listing 8. First, we create a list of all possible categorical values per feature as they are necessary for the `explainer`. Next, we use the `convert_to_lime_format` function[54] adopted from Kevin Lemagnen's Pycon presentation in 2019[68], as the one included in the LIME documentation only works with older versions of Python.

[68]https://speakerdeck.com/klemag/pycon-2019-introduction-to-model-interpretability-in-python

The function converts all existing string variables to integers, so they can be interpreted.

Listing 8: The convert to LIME function

```
# Converts data with categorical values as string to categorical
    values with integers labels.
def convert_to_lime_format(X, categorical_names, col_names=None,
    invert=False):


# If the data is not a dataframe, we need to be able to create one
    if not isinstance(X, pd.DataFrame):
        X_lime = pd.DataFrame(X, columns=col_names)
    else:
        X_lime = X.copy()

    for k, v in categorical_names.items():
        if not invert:
            label_map = {
                str_label: int_label for int_label, str_label in
                    enumerate(v)}

        else:
            label_map = {
                int_label: str_label for int_label, str_label in
                    enumerate(v) }
        X_lime.iloc[:, k] = X_lime.iloc[:, k].map(label_map)

    return X_lime
```

The explainer itself is included in the LIME library and displayed in Listing 9. We set all parameters manually, as the explainer does not possess any default values. First, we call our now formatted dataset and set the mode to classification. Then we give a list of all features in our dataset (line 3) and with `categorical_names=categorical_names` (line 4) we specify which of the variables are categorical. `Categorical_features` (line 5) list the index of all features with a categorical type and `discretize_continuous` (line 6) is a mathematical function that simply helps to produce a better output by converting continuous attributes to nominal attributes. The final parameter, `random_state`, brings consistency into the function, as if not specified it always picks a different number whenever we reload the function.

Listing 9: The LIME tabular explainer

```
explainer = LimeTabularExplainer(convert_to_lime_format(X_train,
    categorical_names).values,
                                 mode="classification",
                        feature_names=X_train.columns.tolist(),
                        categorical_names=categorical_names,
                        categorical_features=categorical_names.keys(),
                        discretize_continuous=True,
                        random_state=42)
```

Explaining an observation We call one observation on which we apply the interpretability framework and subsequently print the classification each model gives for this instance as well as the true label.

We can now convert the output to the LIME format, saving it in the observation variable before defining a standard predict function. The `custom_predict_proba` function, displayed in Listing 10, is able to transform very simple models but also more complex input. It converts the data so it is applicable to the `LIMETabularExplainer`, which we carry out for every model we wish to interpret. After that we can apply the LIME framework to our classification models.

Listing 10: The predict_proba converter

```
# The custom_predict_proba function makes data applicable to the
    respective model

def custom_predict_proba(X, model):
    X_str = convert_to_lime_format(X, categorical_names,
        col_names=X_train.columns, invert=True)
    return model.predict_proba(X_str)

lr_predict_proba = partial(custom_predict_proba, model=lr_model)
dt_predict_proba = partial(custom_predict_proba, model=dt_model)
rf_predict_proba = partial(custom_predict_proba, model=rf_model)
xgb_predict_proba = partial(custom_predict_proba, model=xgb_model)
```

To create a LIME output, we use the logistic regression as an example as shown in Listing 11. Therefore, we define the explanation as `explainer.explain_instance` and include the observation we chose above,

adding the `lr_predict_proba` and five features as this shows us the factors considered the most influential on predicting the target variable.

Listing 11: Calling the LIME output

```
# The explainer applied to the logistic regression

explanation = explainer.explain_instance(observation,
    lr_predict_proba, num_features=5)

explanation.show_in_notebook(show_table=True, show_all=False)
```

Running the code presents us with the LIME output, displayed in Figure 12, consisting of three parts: the prediction probabilities on the left side, the feature probabilities in the center and the feature-value table on the right. The prediction probabilities graph shows the model's decision on that instance, meaning which outcome it predicts and the corresponding probability. In our example it predicts, that it is not going to rain with 83% probability, represented by the blue bar with the number 0 and that it is going to rain with 17%, represented by the orange bar with the number 1. The feature probabilities graph gives insight into how much a feature influences the given decision. For this observation the variable *WindGustSpeed* is the most influential factor and supports the prediction, that it is not going to rain tomorrow. The second most important feature is *Humidity3pm* which weights towards that it is going to rain tomorrow, represented by the number 1. In this case, we display the top five features in our output, but theoretically all the features could be listed that way, ordered by their importance. The last graph is the feature-value table, which also sorts the features by importance, but instead of showing their weight, is given the actual value that this feature possesses in this observation. For example, the fifth feature, *MaxTemp*, shows 21.50 in this table, representing 21.5 degrees Celsius, the maximum temperature on the day of the observation. It is coloured blue, as it is influencing the model's decision towards no rain.

5.3 Evaluating the models on a global level

To analyse the LIME output on a more global level, we apply the framework on fifty observations and aggregate the output in an excel file to compare the graphs with each other. As we analyse four models, we end up with 200

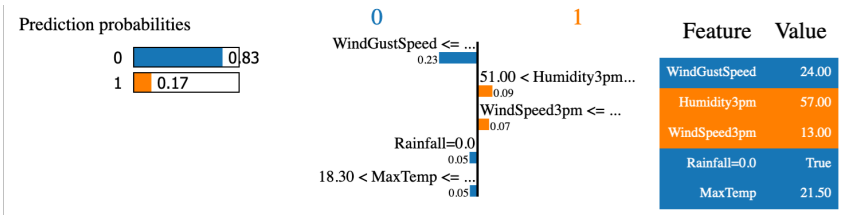

Figure 12: Example LIME output using logistic regression

interpretations in total.

Feature occurrence and influence LIME allows us to look at individual features in more detail and evaluate their influence. In our analysis the framework displays the top five features per observation resulting in 200 total feature counts and 50 top positions per model. Out of this set, *Humidity3pm* occurs most frequently, except for the XGBoost where it is ranked second after *Pressure9am*. It appears 50 times in the analysis of the decision tree and logistic regression, 42 times at the random forest and 48 times at the XGBoost. Furthermore, *Humidity3pm* is not only the most frequent, but is also considered the most important feature, as for the logistic regression it is the most influential feature, meaning it is ranked number one, in all 50 cases and for the decision tree in 47 cases. In case of the random forest, its prediction it is not going to rain is heavily influenced by `Rainfall`, as whenever it did not rain, it is ranked in first or second position, which happens in 22 and 11 cases, respectively. Nevertheless, *Humidity3pm* is also important for the random forest and occurs in 21 cases on the first rank. In the XGBoost classification *Humidity3pm* is considered the most important feature 38 times.

The least considered features are *WindGustDir*, *RainToday* and *Temp9am*, with an occurrence of five, seven and eight times, respectively, which are never ranked within the first or second position.

Considering this values, we now know that *Humidity3pm* is highly predictive for our models, bringing us a step closer to developing a usable application.

The extent of misclassification By displaying the intervals of its classification, LIME enables us to evaluate the accuracy of a single prediction. In terms of a false assessment we calculate the absolute difference between the probabilities assigned to the target variables, measured in percentage points. This tells us by how much the prediction is wrong and results in another

indicator to assess the models. The false classifications are divided into two categories: a wrong prediction with less than 20 percentage points of absolute difference is called a close miss and a prediction with 20 percentage points or over more absolute difference is called a far miss.

The analysis of all observations results in the following: the decision tree classifies 12 out of 50 instances incorrectly, which are split evenly between close and far misses. The average absolute difference of all wrong classifications is 23 percentage points. In terms of the amount of incorrect classifications the logistic regression performs better than the decision tree, with eight wrong classifications, of which five are a close and three are a far miss. In absolute difference the logistic regression performs slightly worse, with around 26 percentage points. The random forest misclassifies nine times, of which six are close and three are far misses and gives us an average of 15 percentage points. Lastly, the XGBoost predicts incorrectly only four times, one time causing a close miss and three times a strong miss, resulting in around 38 percentage points absolute difference.

Considering the different evaluations we conducted, the XGBoost is superior in the majority of cases. With the highest accuracy of 85%, weighted classification report scores of 85% *precision*, 85% *recall*, 84% *f1-score*, a *ROC-test-baseline* of 88% and the least amount of incorrect classifications, it delivers better performance than the other models.

6 A qualitative assessment of LIME

The assessment of LIME's interpretability and usability is split into two parts: firstly, we perform interviews to get an impression of how LIME is interpreted by people who are not familiar with the concept of explainable AI; Secondly, we apply a user experience evaluation framework in order to perform a self assessment of LIME's usability based on its criteria.

6.1 The interviews

We interviewed six people, equally split between male and female, three with prior knowledge of machine learning and three with no prior knowledge. Non of them were familiar with the concept of xAI before participating in the interview. The participants were either academics or in the process of pursuing a degree. In each interview we wanted to find out how interpretable the LIME output is for a person who never worked with xAI before. An overview of the interview results is displayed in Table 10.

The interview was split into two sections, each of them started with an explanation from the interviewer. In the first part the interviewees were given a quick introduction into the topic of the thesis, how to do rain prediction, as well as a quick introduction into machine learning. Then they were shown the first `LIMETabularExplainer` output graph Figure 13 and were asked to assess it based on four questions.

When asking the participants to describe what they see when looking at the LIME output, all interviewees expressed uncertainty about what the illustrations show. All started with identifying the three graphs and tried to make sense of the different numbers. Although a few participants struggled with the prediction-probabilities and feature-value graph, every participant had difficulties interpreting the feature probabilities as the numbers did not seem to add up and there was too much information given in a badly structured way.

People without prior machine learning knowledge struggled to see the relation between the prediction probabilities and the classification, but those with prior knowledge in machine learning concluded, that there is a connection between the feature probabilities and the prediction probabilities graph. Two concluded correctly, that the second smaller numbers on the central graph are probabilities, as they are between 0 and 1 and influence the attribute of the predictability.

When we asked if the participants understand *"why the model made this prediction, and if yes, how?"*, only one out of six answered correctly, that

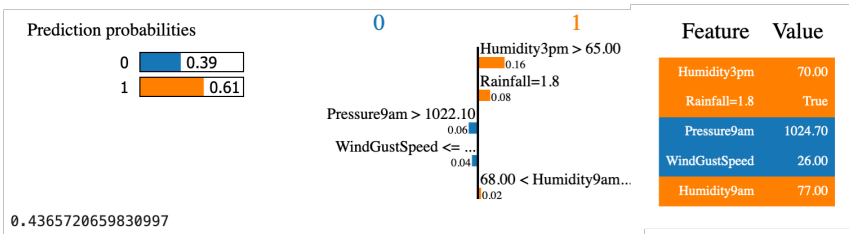

Figure 13: Output of the first part

Participant	ML knowledge	Gender	Illustration	Prediction	Rating part I	Understand part II	Rating part II	R2
1	yes	m	yes	yes	3	improved	8	
2	no	f	no	no	3	improved	6.5	x
3	no	m	no	no	4	improved	7.5	x
4	yes	m	yes	yes	5	improved	9.5	
5	no	f	no	no	4	improved	7.5	x
6	yes	f	no	no	3	improved	7	x

Table 10: Participants' understanding of the LIME output

the classification is determined by the numbers of the feature probabilities graph.

On a scale from 1 to 10, with 1 being the worst and 10 the best, the interpretability of the LIME output was rated with an average of 3.66. The rating between the subgroups differed only slightly, the participants without prior knowledge gave an average of 3.33 and the participants with prior knowledge 4.0, respectively.

The second section started with a short explanation of each graph of the LIME output as well as an explanation of the meaning of the r-squared value at the bottom of the output. Then they were shown Figure 14 and were asked four more questions.

As soon as the participants were given an explanation for each graph the answers improved significantly. Four understood the graphs correctly, but were still uncertain where the probabilities of the prediction probabilities graph resulted from. Two of the participants with a machine learning

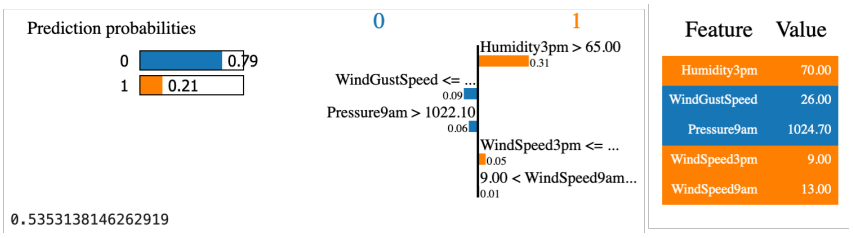

Figure 14: Output of the second part

background understood the framework after the explanation. Another four pointed out that the r-squared scores of both models were low, which resulted in concerns about the reliability of the prediction. Nevertheless, the explainability of the graph after the explanation improved significantly, to an average of 7.66, while participants with prior machine learning knowledge again rated it slightly higher with an average of 8.16 and the other group on average one point lower.

Additionally, five out of six participants stated that the central graph was not very interpretable and three mentioned that they found the choice of colours disturbing. Furthermore, five interviewees suggested a legend, titles or a short explanation should be included in the output visualisation to improve its interpretability.

To sum up, while one can assume that a framework can not be completely understood without any further explanation, the interview results show that the illustration needs some clear improvements to become understandable to a person not familiar with xAI. While the participants with a background in machine learning understood the method quicker, they still struggled severely with LIME's usability.

6.2 Self assessment of the usability

To assess LIME's user experience, we adopt the definition of usability proposed by the International Organisation for Standardisation (ISO)[69] in their ISO 9241-11 1998 report [55]. In the report usability is defined as the *"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"* [55]. As this definition is too broad to be directly applied in our evaluation context, we improve its applicability by including 'Usability Meanings and Interpretations in ISO Standards' [56] by Abran et al. 2003

[69]https://www.iso.org/home.html

and 'New ISO Standards for Usability, Usability Reports and Usability Measures' [57] by Bevan et al. 2016, which both interpret and extend the ISO definition.

6.2.1 Effectiveness

In [57], Bevan et al. state that *"effectiveness has been associated with completing a task completely and accurately, but it is also important to take account of the potential negative consequences if the task is not achieved correctly"*. From this we extract the factors measure of completion, measure of accuracy and negative consequences to rate effectiveness.

Furthermore, in [56] the question *"How well do users achieve their goal using the system?"* is used to rate the overall interpretability. When we apply the inputs of Bevan et al. and Abrans et al. to LIME, we end up with following questions which can be used to assess effectiveness:

Q1) How effective does LIME achieve model interpretability?

- (a) How complete is the explanation on a local level?
- (b) How complete is the explanation on a global level?
- (c) Could accurate results be misinterpreted?
- (d) What negative consequences arise from a misinterpretation?

(a) LIME is a local explainability framework, therefore it calculates the influence of every feature with the associated feature importance on a local level, which helps to make a model's classification more understandable. Nevertheless, the connection between the prediction probabilities and the feature probability graph is incomplete as currently only the feature importance score is displayed, however these scores do not add up to the prediction probabilities. As displayed in Figure 15, the feature *Humidity3pm* with a feature probabilities score of 0.31 exceeds the total prediction probability of 0.21 that it is going to rain, while the overall classification was in favor of no rain. Many cases show contradictions that can only be explained by assuming, that the displayed prediction probabilities are not the sum of the feature probabilities, but the result of another calculation not obvious to us as a user.

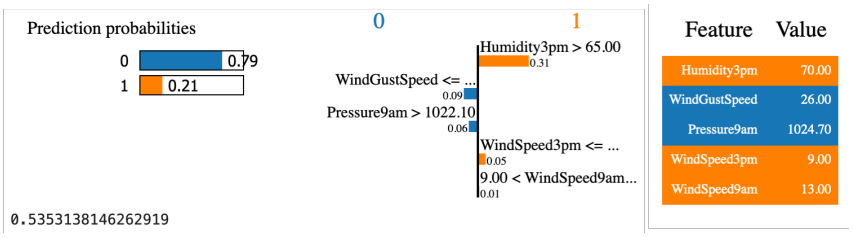

Figure 15: Example of a LIME output

(b) While LIME processes locally, the author has specified the framework's purpose for comparing different model performances with each other [58] which is only possible on a global level. However, the `LIMETabularExplainer` performs much poorer globally, as it does not include a function or interface to allow a global evaluation. Thus, we have to extract several observation outputs manually and analyse them in an Excel file, as we did in the global analysis of section 5. We either recommend implementing performance indicators that allow a global comparison with other models or add a function to extract the local outputs of several random observations as a spreadsheet, so the user can calculate indicators necessary for a global comparison themselves.

(c) The interpretations of the local predictions appear to be accurate. But we see a risk for misinterpretation of the tabular explainer, as no comprehensive explanation of it has been published yet [59]. Therefore, we have to rely on third party explanations like online articles[70] [71] or talks on YouTube[72] [73].

(d) In case of a misinterpretation of the LIME evaluation the severity of the negative consequences depends on the use-case. For example the implication of a wrongly deployed automated defense system as presented in 'DARPA's Explainable Artificial Intelligence Program' [60] and our rain prediction model for Australia, differ. In our case a mistake in the interpretation could lead to a faulty feature importance and therefore a wrong rain forecast, in DARPA's case, a biased object classification by an automated defense systems puts lives at risks.

[70] https://medium.com/analytics-vidhya/explain-your-model-with-lime-5a1a5867b423

[71] https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/

[72] https://www.youtube.com/watch?v=CY3t11vuuOM

[73] https://www.youtube.com/watch?v=C80SQe16Rao

To conclude, an effective model interpretation with LIME can be achieved, as we can assess our models' performances on a local as well as on a global level. However, the process to extract the output, which is necessary to get a complete global understanding, is tedious and requires additional software. Therefore the framework is only effective to some extend.

6.2.2 Efficiency

To evaluate efficiency Bevan et al. [57] identify the following factors: task time, time efficiency, cost-effectiveness, productive time ratio, unnecessary actions and fatigue. We aggregate them to a list with mutually exclusive components and conclude with the question raised by Abran et al. *"What resources are consumed in order to achieve the goal?"* [56].

Q2) What resources are consumed in order to achieve interpretability?

- (a) How much time does it take to use LIME?
- (b) What other costs are involved?
- (c) Does this process cause fatigue?

(a) Both, the time to set up LIME as well as the time to analyse the output play a role in this context. The setup works well, however the `LIMETabularExplainer` setup documentation, which is provided by the author, is depreciated as it relates to several old packages. Therefore, the initial process of applying the original notebook and trying to find workarounds consumed a lot of time. Additionally, the analysis of the LIME output took a considerable amount of time, as the documentation of the graphs is non-transparent as stated in the effectiveness evaluation. On the up-side, the time it takes to compute and display an observation is minimal.

(b) As LIME is an open source tool, no licensing costs are involved and also the publications, documents and videos to understand the tool are free to use.

(c) To apply LIME only on a few observations is a quick process and therefore not costly from a performance perspective. For the global interpretation however it took hours of repetitive manual copying and pasting LIME output from the notebook into an Excel file which was a tedious process. As

no benchmark on the number of observations necessary to evaluate models globally in the documentation and publications about LIME is given, we had no indicator about how many outputs would be sufficient.

While LIME is an open source software, the application of it comes at a cost. A significant amount of time is necessary to make use of the framework. A more detailed documentation, guideline and explanation of the output would improve the efficiency. Especially providing a benchmark of how many observations are necessary to conduct the global analysis would safe the user time and effort.

6.2.3 Satisfaction

Satisfaction is the least standardised of the three parameters as it is highly dependent on the user and use-case [57]. Based on Bevan et al. satisfaction aims to take *"positive attitudes, emotions and/or comfort resulting from use of a system, product or service"* [57] into account. The question Abran et al. raise to assess satisfaction is *"How well does the user feel about the use of the system?"* [56], which we include in our analysis. Combining both ideas we finalise the third and last factor:

Q3) How satisfying is the application of LIME?

- (a) Do we have a positive or negative attitude towards the tool?
- (b) What emotions arise from using it?
- (c) How satisfying is the final result?

(a) At the start of the implementation our attitude was very positive, as LIME's original introduction and publication was very promising in a sense that it helps to interpret and trust a model's prediction. During the setup our attitude deteriorated due to a lack of documentation and support which posed an even bigger problem during the analysis. LIME gives insight into a model's processes, but here again it takes a lot of effort to get a clear understanding of the framework, which has a negative influence on our attitude. Naturally, once one has learned how to apply and interpret LIME, the process is a much more pleasant one.

(b) The lack of a clear and explicit guideline makes understanding LIME a frustrating process. However, reaching the point of a better overall un-

derstanding of our models raises positive feelings. Especially LIME's short processing time to produce the graphs makes it easy to evaluate several instances in a row, which leads to a very pleasant user experience.

(c) The output of the `LIMETabularExplainer` unquestionably helps to understand the model's classification process, as it offers insights conventional methods can not provide, which causes satisfaction. However, this satisfaction could be increased by eliminating doubt about the relationships between the local indicators and offering a global analysis.

When applying LIME for the first time the framework created some frustration and uncertainty. Although it is clear that there is still room for improvement, the framework enables model interpretability which is the desired outcome. Therefore we can express our overall satisfaction of the application of LIME.

7 Conclusion & Future Work

In this chapter we outline how this thesis accomplished to assess LIME's suitability for the interpretation of tabular machine learning model and point out what additional research could be conducted in this field.

7.1 Conclusion

To gain experience with LIME we first applied the `LIMEImageExplainer` to an image classification model, as the setup and analysis of this function is very well documented. We then began the assessment of LIME's tabular interpretation function, by pre-processing a tabular dataset and applying four state of the art machine learning algorithms, namely *decision tree*, *logistic regression*, *random forest* and *XGBoost* on it. We first compared their performance using the classification report and the receiver operating characteristics curve and found that a decision with respect to which model performs best could not be made confidently as we lacked insight into the process. Subsequently we applied the `LIMETabularExplainer` and analysed single observations on a local level and conducted a global evaluation of the four models. As a result, we established that the XGBoost had a superior performance in the majority of cases. Furthermore, we conducted a qualitative analysis of LIME utilizing interviews to assess it's interpretability and developed a usability framework to evaluate our experience using LIME. The interviewees mainly identified the `LIMETabularExplainer` output to be rather confusing as it lacked an easy to understand explanation of how to read it. Our self assessment, which was conducted according to ISO's user experience definition, identified weaknesses in LIME's application, especially as it lacks an extensive and updated documentation on how to apply the framework, which made the process rather frustrating. Overall, we conclude that while LIME helps to increase interpretability of a model processing tabular data, its usability, especially when conducting a global interpretation, can be improved in various ways.

7.2 Future Work

To assess LIME's interpretability of tabular models further, we suggest to repeat the quantitative evaluation several times, selecting different features to train and test our model with, and analyse the difference in feature importance with LIME to get further insight into the model's process. In terms of a qualitative assessment we suggest to evaluate the improvement in user experience by changing the way the LIME output is presented by including a legend explaining the different output graphs or changing the feature

probability to display the actual influence of a feature on the prediction. Furthermore, an extensive comparison of model agnostic local interpretability frameworks could be conducted, using our user experience framework to assess them.

References

[1] P. McCorduck, M. Minsky, O. G. Selfridge, and H. A. Simon, "History of artificial intelligence," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977*, 1977, pp. 951–954. [Online]. Available: http://ijcai.org/Proceedings/77-2/Papers/083.pdf

[2] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," 2017.

[3] N. Papernot and P. McDaniel, "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning," 2018.

[4] M. Hind, D. Wei, M. Campbell, N. C. F. Codella, A. Dhurandhar, A. MojsiloviÄ‡, K. N. Ramamurthy, and K. R. Varshney, "Ted: Teaching ai to explain its decisions," 2018.

[5] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017.

[6] J. Lei, M. GâTMSell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman, "Distribution-free predictive inference for regression," *Journal of the American Statistical Association*, vol. 113, no. 523, pp. 1094–1111, 2018.

[7] G. Plumb, D. Molitor, and A. S. Talwalkar, "Model agnostic supervised local explanations," in *Advances in Neural Information Processing Systems*, 2018, pp. 2515–2524.

[8] M. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 02 2016, pp. 97–101.

[9] S. Mishra, B. L. Sturm, and S. Dixon, "Local interpretable model-agnostic explanations for music content analysis," in *ISMIR*, 2017.

[10] L. Hu, J. Chen, V. N. Nair, and A. Sudjianto, "Locally interpretable models and effects based on supervised partitioning (lime-sup)," *arXiv preprint arXiv:1806.00663*, 2018.

[11] A. Dhurandhar, T. Pedapati, A. Balakrishnan, P.-Y. Chen, K. Shanmugam, and R. Puri, "Model agnostic contrastive explanations for structured data," *arXiv preprint arXiv:1906.00117*, 2019.

[12] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," in *Advances in neural information processing systems*, 2016, pp. 3387–3395.

[13] S. Tan, R. Caruana, G. Hooker, and Y. Lou, "Detecting bias in black-box models using transparent model distillation," *arXiv preprint arXiv:1710.06169*, 2017.

[14] M. T. Ribeiro, S. Singh, and C. Guestrin, "Nothing else matters: model-agnostic explanations by identifying prediction invariance," *arXiv preprint arXiv:1611.05817*, 2016.

[15] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3319–3328.

[16] A.-H. Karimi, G. Barthe, B. Belle, and I. Valera, "Model-agnostic counterfactual explanations for consequential decisions," *arXiv preprint arXiv:1905.11190*, 2019.

[17] S. Sharma, J. Henderson, and J. Ghosh, "Certifai: Counterfactual explanations for robustness, transparency, interpretability, and fairness of artificial intelligence models," *arXiv preprint arXiv:1905.07857*, 2019.

[18] G. Casalicchio, C. Molnar, and B. Bischl, "Visualizing the feature importance for black box models," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2018, pp. 655–670.

[19] R. Khanna, B. Kim, J. Ghosh, and O. Koyejo, "Interpreting black box predictions using fisher kernels," *arXiv preprint arXiv:1810.10118*, 2018.

[20] N. Puri, P. Gupta, P. Agarwal, S. Verma, and B. Krishnamurthy, "Magix: Model agnostic globally interpretable explanations," *arXiv preprint arXiv:1706.07160*, 2017.

[21] G. J. Katuwal and R. Chen, "Machine learning model interpretability for precision medicine," *arXiv preprint arXiv:1610.09045*, 2016.

[22] J. Singh and A. Anand, "Exs: Explainable search using local model agnostic interpretability," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 770–773.

[23] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," *arXiv preprint arXiv:1805.10820*, 2018.

[24] M. R. Zafar and N. M. Khan, "Dlime: a deterministic local interpretable model-agnostic explanations approach for computer-aided diagnosis systems," *arXiv preprint arXiv:1906.10263*, 2019.

[25] T. Peltola, "Local interpretable model-agnostic explanations of bayesian predictive models via kullback-leibler projections," *arXiv preprint arXiv:1810.02678*, 2018.

[26] S. García, A. Fernández, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, p. 959, 2009.

[27] D. P. Green and H. L. Kern, "Modeling heterogeneous treatment effects in large-scale experiments using bayesian additive regression trees," in *The annual summer meeting of the society of political methodology*, 2010.

[28] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008.

[29] J. Singh and A. Anand, "Model agnostic interpretability of rankers via intent modelling," in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020, pp. 618–628.

[30] L. Arras, F. Horn, G. Montavon, K.-R. Müller, and W. Samek, """ what is relevant in a text document?""": An interpretable machine learning approach," *PloS one*, vol. 12, no. 8, 2017.

[31] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. MÃžller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, no. Jun, pp. 1803–1831, 2010.

[32] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[33] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE*

conference on computer vision and pattern recognition, 2016, pp. 2921–2929.

[34] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3429–3437.

[35] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *Advances in Neural Information Processing Systems*, 2017, pp. 6967–6976.

[36] P. Cortez and M. J. Embrechts, "Opening black box data mining models using sensitivity analysis," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. IEEE, 2011, pp. 341–348.

[37] S. Lundberg and S.-I. Lee, "An unexpected unity among methods for interpreting model predictions," *arXiv preprint arXiv:1611.07478*, 2016.

[38] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "L-shapley and c-shapley: Efficient model interpretation for structured data," *arXiv preprint arXiv:1808.02610*, 2018.

[39] C. Frye, I. Feige, and C. Rowat, "Asymmetric shapley values: incorporating causal knowledge into model-agnostic explainability," *arXiv preprint arXiv:1910.06358*, 2019.

[40] O. Bastani, C. Kim, and H. Bastani, "Interpretability via model extraction," *arXiv preprint arXiv:1706.09773*, 2017.

[41] J. J. Thiagarajan, B. Kailkhura, P. Sattigeri, and K. N. Ramamurthy, "Treeview: Peeking into deep neural networks via feature-space partitioning," *arXiv preprint arXiv:1611.07429*, 2016.

[42] M. Ribeiro, "Tutorial - image classification keras," accessed: 2020-04-18. [Online]. Available: https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%20-%20Image%20Classification%20Keras.ipynb

[43] F. Chollet, "Tutorial - image classification keras," accessed: 2020-04-18. [Online]. Available: https://keras.io/getting_started/intro_to_keras_for_engineers/

[44] "Wikipedia," accessed: 2020-04-18. [Online]. Available: https://dictionary.cambridge.org/de/worterbuch/englisch/boathouse

[45] aaronburden, "Alanson," Oct 2018. [Online]. Available: https://unsplash.com/photos/KyWTf39MX6U

[46] MWanner, "Wikipedia," https://creativecommons.org/licenses/by-sa/3.0/, Oct 2007. [Online]. Available: https://en.wikipedia.org/wiki/Boathouse#/media/File:Topridge_Boathouse.jpg

[47] C. Molnar, *Interpretable Machine Learning*, 1st ed. Christop Molnar, 2020, https://christophm.github.io/interpretable-ml-book/.

[48] S. Hara and K. Hayashi, "Making tree ensembles interpretable: A bayesian model selection approach," 2016.

[49] J. E. T. Akinsola, "Supervised machine learning algorithms: Classification and comparison," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 48, pp. 128 – 138, 06 2017.

[50] D. L. Streiner and J. Cairney, "What's under the roc? an introduction to receiver operating characteristics curves," *The Canadian Journal of Psychiatry*, vol. 52, no. 2, pp. 121–128, 2007. [Online]. Available: https://doi.org/10.1177/070674370705200210

[51] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[52] J. Surowiecki, *The Wisdom of Crowds: Why the Many are Smarter than the Few*. Little Brown, 2004.

[53] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[54] K. Lemagnen, "helpers.py," https://github.com/charlespwd/project-titlehttps://github.com/klemag/PyconUS_2019-model-interpretability-tutorial/blob/master/helpers.py, 2019.

[55] I. O. for Standardisation, "Iso 9241-11:1998(en) ergonomic requirements for office work with visual display terminals (vdts) â" part 11: Guidance on usability," 1998.

[56] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability meanings and interpretations in iso standards," *Software Quality Journal*, vol. 11, pp. 325–338, 11 2003.

[57] N. Bevan, J. Carter, J. Earthy, T. Geis, and S. Harker, "New iso standards for usability, usability reports and usability measures," vol. 9731, 07 2016, pp. 268–278.

[58] M. T. Ribeiro, S. Singh, and C. Guestrin, "Model-agnostic interpretability of machine learning," *arXiv preprint arXiv:1606.05386*, 2016.

[59] M. T. Ribeiro, "Lime tabular package," 2016. [Online]. Available: https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_tabular

[60] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (xai) program," pp. 44–58, Jun 2019. [Online]. Available: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2850