MASTER THESIS

Model-Agnostic Black Box Explainers For Various Data Types

for the degree of
Master of Science

University of Trier
Faculty IV

submitted by

Artem Smirnov
Master's Programme Data Science
Matr.-Nr. 1604715

Supervisor: Prof. Dr. Volker Schulz
Second examiner: Dr. Stephan Schmidt

Trier, February 2, 2024

ERKLÄRUNG ZUR MASTERARBEIT

Hiermit erkläre ich, dass ich die Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.
Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

DECLARATION FOR THE MASTER'S THESIS

I hereby declare that I have written this Master's thesis independently and that I have not used any other sources and aids other than those indicated and that the thoughts taken directly or indirectly directly or indirectly from external sources.
The master thesis has not been submitted to any other examination office in the same or a comparable form. It has also not been published to date.

Datum	Unterschrift

Contents

1 Lists of Abbreviations, Algorithms, and Figures **2**

2 Introduction **4**

3 Literature review **6**

4 Data analysis and used machine learning models **8**
4.1 Data description 8
4.1.1 Tabular datasets 8
4.1.2 Imagery dataset 8
4.2 Image Segmentation using SLIC 9
4.3 Tabular data. Random forests. 10
4.3.1 Method description 10
4.3.2 Random forest classifier. Breast cancer data. 12
4.3.3 Random forest regression. California housing data. 15
4.4 Imagery data. Convolutional Neural Networks. 17
4.4.1 Method description 17
4.4.2 Implementation of Convolutional Neural Networks 20

5 Black box explainers **25**
5.1 LIME 25
5.1.1 Implementation of LIME for tabular data 28
5.1.2 Implementation of LIME for imagery data 31
5.2 SHAP 34
5.2.1 Implementation of SHAP for tabular data 37
5.2.2 Implementation of SHAP for imagery data 40

6 Conclusions **43**

7 Bibliography **45**

8 Appendix **49**

1 Lists of Abbreviations, Algorithms, and Figures

List of Figures

1 Example of the RGB color model for the sign “Priority roads.” The central part is shown as RGB values for 3 filters (red, green, and blue respectively). 9
2 Median house values in Californian districts and the forecast. 16
3 Example of convolution for the real image from the trains set. 18
4 Convoluted image. The key features (letters, sign shape) became more distinct. . . 18
5 Max-pooling of the convoluted image. 19
6 The framework of the CNN. Initial framework: [Mov]; visualization is based on: [Rat21]. 20
7 Accuracy and loss of the final CNN classifier. Blue line: training data, red line: validation data. Possible cause of the discrepancy: dropout mechanism (see explanations above). 23
8 LIME explanations for the classifier based on breast cancer dataset. Malignant case. 29
9 LIME explanations for the classifier based on breast cancer dataset. Benign case. . . 29
10 LIME explanations for the test instance of the RF regression based on California housing dataset. The real house price equals 2.186. 30
11 Selected picture to be classified with CNNs and explained with LIME and SHAP. . 31
12 Top-5 predictions for the sign 30km speed limit. Predicted probabilities for the wrong classes are close to 0. 31
13 Segmented image with 100 superpixels to be used in the further analysis. 32
14 Randomly selected picture, which is similar to the initial one 11. 32
15 Matrix, which shows randomly kept (lighter color) and hidden (darker color) superpixels for the first 10 instances. Here columns indicate number of a superpixel, which can be either kept or hidden. The first row is the initial image without any hidden parts. 32
16 Top 10 LIME explanations by importance for the class “30km”, the correct label. . 33
17 Top 10 LIME explanations by importance for the class “80km”, the wrong label. . . 33
18 SHAP for the test benign case. The prediction is correct. 37
19 SHAP for the test malignant case. The prediction is correct. 37
20 SHAP for the global explanations of the random forest classifier. 38
21 SHAP for the test instance. The real house price equals 2.186. 38
22 Comparison of real and predicted Californian housing prices, where SHAP values for both the latitude and longitude are indicated. 39
23 SHAP for the global explanations of the random forest regression. 39
24 SHAP for the segmented image (SLIC). Classes are arranged in descending order of likelihood, starting with the most likely on the left, which is “30km”. 40
25 SHAP for the initial image, i.e. without any segmentation. 41
26 SHAP for the first 5 test images. The case of linear segmentation. 41
27 Results of the decision tree classifier. 54
28 Results of the decision tree regression. Only a fragment is shown due to the model depth which equals to 10. 55

List of Algorithms

1 Pseudocode for SLIC 10
2 Pseudocode for the random forest (for both regression and classification cases) . . . 12
3 Pseudocode for LIME for tabular data 27
4 Pseudocode for LIME for imagery data 28
5 Pseudocode for SHAP (applicable for both tabular and imagery data) 35

List of Abbreviations

1. **CIELAB** Commission Internationale de l´Eclairage; 'L' stands for lightness, 'A' is the green–red axis, and 'B' is the blue–yellow axis.
2. **CNN** Convolutional Neural Network
3. **FN** False Negative
4. **FP** False Positive
5. **GTSRB** German Traffic Sign Recognition Benchmark
6. **LIME** Local Interpretable Model-Agnostic Explanations
7. **MAE** Mean Absolute Error
8. **MAPE** Mean Absolute Percentage Error
9. **ML** Machine Learning
10. **MSE** Mean Square Error
11. **NN** Neural Network
12. **OLS** Ordinary Least Squares
13. **ReLU** Rectified Linear Unit
14. **RF** Random Forest
15. **RGB** Red Green Blue
16. **RMSE** Root Mean Square Error
17. **RMSprop** Root Mean Squared Propagation
18. **SHAP** SHapley Additive exPlanations
19. **SGD** Stochastic Gradient Descent
20. **SLIC** Simple Linear Iterative Clustering
21. **TN** True Negative
22. **TP** True Positive
23. **UCI ML** University of California Irvine machine learning repository
24. **XAI** Explainable Artificial Intelligence

2 Introduction

Nowadays machine learning plays a significant role in numerous spheres of life. ML is constantly evolving, researchers create and improve procedures and frameworks in data analysis. Deep neural networks, including convolutional neural networks, and ensemble algorithms such as random forests are no longer something exotic in statistical analysis. These techniques have already changed business processes in medical care, agriculture, finance, car industry, and many others.

However, as of now, these approaches exhibit some imperfections. There is something apart from some hesitations about their imperfections like under/over-fitting, high computational complexity, powerful hardware requirements. Of course, these issues take place, and we should not forget them while modeling. Despite that, it is crucial to address another serious problem and potential jeopardy behind the use of machine learning models.

To begin with, classical statistical (or econometric) models based on the methods of least squares or maximum likelihood are self-explainable. This holds true for the decision tree method and other simple algorithms as well. For instance, there is a task to predict a risk to get the lung cancer. There is some dataset consists of some patients' characteristics such as their age, genetic predisposition, whether they smoke or not, if so, frequency of smoking, some other habits, whether they work with harmful chemicals or not, key area quality indicators in their city or neighborhood, etc. We may use these exogenous factors to assess the likelihood to get some type of tumor. At the final step, if we use standard methods such as probit regression, we receive p-values for the variables. Thus, we can get the info about the feature and overall model statistical significance. Also, we may look into margin effects (for probit, logit e.g.), which show the direction of change in the probability of getting a disease from increase or decrease of the particular parameter. In case of continuous dependent variable, simple linear models such as OLS can be tried first. This method is even more transparent. Here we are allowed to interpret the regression coefficients directly.

Nonetheless, internal mechanisms behind any prediction of most machine learning models are opaque and they operate as a "black box". It means that our knowledge of an obscured structure of a particular neural network or other complicated ML models remains imprecise and vague. The inner framework of an arbitrary artificial neural network definitely contains a substantial number of weights for neurons, which are used while both training and prediction for the new observations. However, we cannot explain how it really works for the particular case. These weights without any context say nothing about interconnections between independent variables and an endogenous factor to be predicted. The feature importance in this case remains undetermined. So, the use of deep learning and ensemble models comes with risks of missing something fundamental. This is understanding of the principles of how a particular forecast was made. The broader question here is: "Why should we trust the model?" This question should be raised in all cases of using ML, with a particular emphasis on crucial areas such as car industry or medicine. Unfortunately, there are many research papers that overlook the real need for additional verification of resulting models.

This work is aimed to provide some most typical examples of ML usage and its a posteriori black box interpretations. To perform the second part, we will need to use some surrogate methods, namely SHAP and LIME. Although they are not flawless and have some limitations, these algorithms are some of the most popular and well-proven in practice. Hence, Explainable AI approach will be

implemented, which is considered to be an important area and emerging technology domain in many countries. For instance, European Commission proposed the EU AI Act[23]. AI systems are categorized based on the level of risk, and if it is e.g. high, the result of these systems shall be explainable and transparent [21].

The peculiarity of this work is delving into LIME and SHAP algorithms under conditions where detailed descriptions are partially lacking, both in theoretical articles and documentation of packages in Python. As for now, full and systematic guide for these methods does not seem to exist, so there were analysed and summarized several theoretic studies and the code of statistical packages. The objective of the work was to provide comprehensive explanations and hands-on examples of how Explainable Artificial Intelligence (XAI) techniques operate across various types of data. There were selected two most used entities: tabular (cross-sectional) and imagery data.

3 Literature review

This part can be split into two sections: theoretical studies and practical articles. The materials included here focus solely on black box explainers and adjacent areas. So all necessary sources for the correct use of the ML techniques will be indicated in the text of the work itself.

The list of theoretical literature includes Interpretable machine learning by Christoph Molnar, 2020[Mol22]. This popular guide contains contains the necessary basic concepts of LIME and SHAP in a concise format. The author provided some real examples which are simple enough to get the general idea how XAI methods work. The researcher included most essential theoretical explanations for SHAP. Molnar also suggested a pseudocode for this method. However, this paper is not all-encompassing, and the current paper suggests to use another pseudocode for SHAP to make it more clear. It is also worth mentioning that there is only a single general formula in the LIME section, so it can be uninformative to understand how the method actually works. This thesis intends to examine and explain the algorithms of LIME and SHAP and their application to various types of data in detail.

Initially, LIME was suggested in "Why Should I Trust You?" Explaining the Predictions of Any Classifier[Gue16]. Authors introduced "explanation technique that explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction." The researchers provided algorithm description and a pseudocode for better understanding. They were focused on classification of imagery data and illustrated the predictions. So, there were added "dissected" image, which contained several objects to be classified (an electric guitar, an acoustic guitar, and a labrador retriever). According to LIME, in this case the model classified objects correctly respectively their shape and other patterns.

SHAP was introduced by Lundberg and Lee[Lee17]. The classical method based on Shapley value implies that all variables should be taken into analysis, so it demands a substantial amount of computing time. Lundberg and Lee modified this algorithm, so that it could provide explanations for the case with fewer features too. The experimental results demonstrate that SHAP can outperform its alternatives, including LIME. Thus, the explanations made by SHAP are intuitive and comprehensible to humans. Some more profound insights from the theoretical perspective will be described later.

However, practical materials were also used while studying this topic. I prefer to start with the review article on both LIME and SHAP[Lek23]. This commentary discussed the importance of XAI methods in making machine learning models more transparent and trustworthy. It was focused on two commonly used black box explainers, specially in the context of tabular data. The commentary explores how these methods generate explainability metrics and presents a framework for interpreting their outputs, emphasizing their respective pros and cons. The authors intended to explain the results of the decision tree, light gradient-boosting machine, logistic regression, and support vector machines classifier. Based on previous papers and the real experience, the researchers concluded that SHAP and LIME were XAI methods used to understand machine learning models across various fields, especially in sensitive domains. However, data scientists often trust XAI explanations but misapply interpretability tools and struggle to understand visualized outputs. To enhance understanding, it is important to present SHAP results with clear language and consider model assumptions. Comparing SHAP outcomes from different models, using normalized

movement rate (NMR) values for global model explanation, and applying approximated SHAP for local explanations can improve interpretability. Converting feature scores into more understandable forms, such as coefficient values, can also enhance comprehension. Then, they come to the solution that LIME was valuable because it explained local model linearity in simple terms, helping users who may not be familiar with the concept. Notwithstanding, it is essential to note that LIME's explanations can vary for different instances, making it too specific to a single subject and not a general interpretation for the entire model. I opted to incorporate both XAI methods to conduct a real-world comparison between them.

Another empirical article was dedicated to the use of LIME and SHAP for CNNs[Mez23]. I used the similar framework in the further sections. The authors intended "to increase companies' motivation for using ML." The researchers suggested methodology flowchart of the model selection based on XAI results. Stadlhofer and Mezhuyev analyzed industrial imagery data. So, the dataset included two classes: "good" and "defect" products. After getting CNN predictions they used LIME and SHAP and compared these two approaches. As a result, algorithms demonstrate proficient performance. However, LIME is less stable, but requires less computational time.

4 Data analysis and used machine learning models

To show how XAI methods work for various data types, tabular and imagery cases are used here.

4.1 Data description

4.1.1 Tabular datasets

Here we work with cross-sectional data. There are two different cases. They were used to show that SHAP and LIME suit well for interpreting of both regressions and classifiers. These datasets will be used to train both random forest models and decision trees, with a particular focus on interpretation.

Regression case. For this part I used California housing dataset. It was taken from **Scikit**, a Python package. The total number of samples is 20640, and a response variable is the median house value for California districts. The attributes are:

1. MedInc median income in block
2. HouseAge median house age in block
3. AveRooms average number of rooms
4. AveBedrms average number of bedrooms
5. Population block population
6. AveOccup average house occupancy
7. Latitude house block latitude
8. Longitude house block longitude

Classification case. This section included the use of a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset[Scib], which is also uploaded from **Scikit**. This dataset contains 569 observations and 30 independent factors such as worst concavity and mean concave points of a breast tumor for instance. The target variable (or responding variable) contains **212 malignant** and **357 benign** cases. The full list of regressors is in Appendix 1.

4.1.2 Imagery dataset

This dataset is taken from GTSRB - German Traffic Sign Recognition Benchmark Multi-class[Myk]. The data consists of PNG photos of **43 various traffic sign types**. The full list of traffic sign labels is in Appendix 2. The training set contains **39209** images and there are **12630** pictures in the test one. The pictures are of RGB standard, which means that there are 3 channels used - the red, green, and blue colors. Each of them is a set of integers in $[0, 255]$, where 0 stands for the darkest shade, and 255 represents the lightest one1.

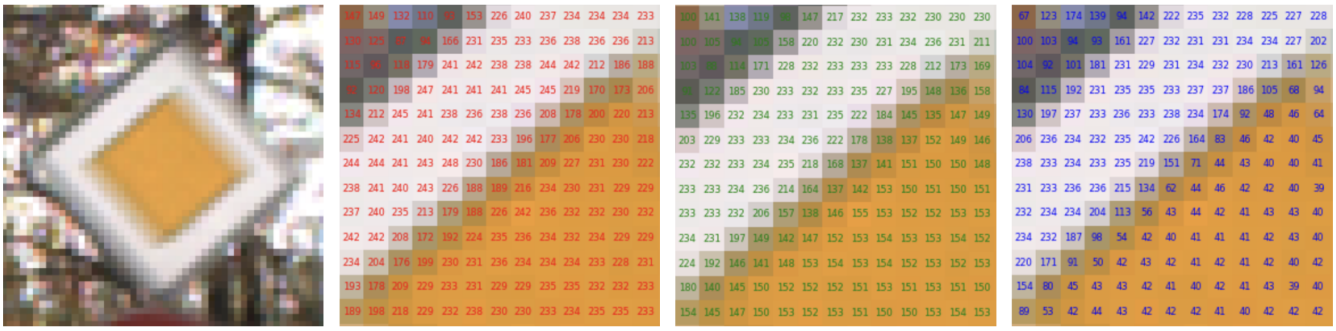

Figure 1: Example of the RGB color model for the sign "Priority roads." The central part is shown as RGB values for 3 filters (red, green, and blue respectively).

4.2 Image Segmentation using SLIC

This part is dedicated to handling the imagery data while using LIME and SHAP. Researchers generally do not analyze the impact of particular pixels on the overall prediction. More often, they deal with segments or, as they are also called in the literature, **superpixels**. This procedure is similar to the clustering, although we need to adapt these methods to working with images. For this purpose, it was decided to use one of the popular techniques. It is called SLIC[Süs12] [Irv16] (Simple Linear Iterative Clustering), and this method is implemented as **slic** in Skimage, a Python package. The `compactness` parameter means trade-off color-similarity and proximity. The optional factor of `n_segments` denotes the number of centers for k-means. This algorithm simply performs k-means clustering in the space of color information and image location. Since SLIC is one of the simplest methods of image segmentation, it is faster than other options and can remain efficient as well. One can find the algorithm of SLIC below, this pseudocode is provided in [Süs12]. In this article authors say that "the approach generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. This is done in the 5-dimensional [labxy] space, where [lab] is the pixel color vector in CIELAB color space, which is widely considered as perceptually uniform for small color distances, and xy is the pixel position." This algorithm will be used in the further parts.

Algorithm 1 Pseudocode for SLIC

input : 5D-image (3 colors and 2 axes: x, y), *threshold*
input : K of approximately equally-sized superpixels
input : for an image with N pixels, the size of each superpixelpixel is N/K pixels
input : superpixel center at every grid interval $S = \sqrt{N/K}$, $k = [1, K]$
output : clusters
initialize: cluster centers $C_k = [l_k; a_k; b_k; x_k; y_k]^T$ by sampling pixels at regular grid steps S.
while $E \leq threshold$ **do**
    for each cluster center C_k **do**
        Assign the best matching pixels from a $2S \times 2S$ square neighborhood around
        the cluster center according to the distance measure:
        $d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$
        $d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$
        $D_s = d_{lab} + \frac{m}{S} d_{xy}$, where m is a constant, in the article it equals to 10
    end for
    Compute new cluster centers and residual error E {$L1$ distance between previous
    centers and recomputed centers}
end while
Enforce connectivity.

4.3 Tabular data. Random forests.

4.3.1 Method description

The random forest classifier was initially described in 1995[Ho95]. The logic of its use was driven by issues and limitations of decision trees. The author comes to the conclusion: "trees derived with traditional methods often cannot be grown to arbitrary complexity for possible loss of generalization accuracy on unseen data. The limitation on complexity usually means suboptimal accuracy on training data." Thus, the random subspace method has found the application in enhancing decision trees, and the resulting ensemble models based on decision trees are commonly known as random forests now. Random forest performs a random selection of a subset of variables to use to build each decision tree model.

There is a definition[Bre01] of **a random forest classifier**: "a random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, ...\}$ where the $\{\Theta_k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x." In elucidating the motivation behind using random forests, which involve a multitude of trees to prevent overfitting (a typical issue with decision trees), the researcher[Bre01] introduces a theorem. It starts with the definition of the margin function under "given an ensemble of classifiers $h_1(x), h_2(x), ..., h_K(x)$":

$$mg(X, Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j) \tag{1}$$

where "$I(\cdot)$ is the indicator function". The training set is being "drawn at random from the distribution of the random vector Y, X". Then, according to this article, the generalization error can be defined as follows:

$$PE^* = P_{X,Y}(mg(X, Y) < 0) \tag{2}$$

So, in this paper there stated that "as the number of trees increases, for almost surely all sequences $\Theta_1, ...$, PE^* converges to the formula 3." It is called Theorem 1.2 in this paper, and the proof was attached (see Appendix 3).

$$P_{X,Y}(P_\Theta(h(X,\Theta) = Y) - max_{j \neq Y} P_\Theta(h(X,\Theta) = j) < 0) \tag{3}$$

To sum up, this theorem explains "why random forests do not overfit as more trees are added, but produce a limiting value of the generalization error."

There is also **a random forest regression**, and there is a definition[Cre20]: "random forest (RF) models are machine learning models that make output predictions by combining outcomes from a sequence of regression decision trees." As can be noticed, it is an analogue of the classifier method, and most inferences are identical. The most significant difference is dealing with numerical values instead of class labels. Nevertheless, we should update the PE^*4 and provide an additional explanation for why a random forest regression can outperform an ordinary decision tree is presented in Theorem 11.2. So, the author writes "assume that for all Θ, $EY = E_X h(X,\Theta)$, then:"

$$PE^* = E_{X,Y}(Y - h(X))^2 \tag{4}$$

$$PE^*(forest) \leq \rho PE^*(tree) \tag{5}$$

"where ρ is the weighted correlation between the residuals $Y - h(X,\Theta)$ and $Y - h(X,\Theta^{'})$ where $\Theta, \Theta^{'}$ are independent." The proof is in Appendix 4.

Finally, this method normally outperforms the accuracy of decision trees. Although random forest lacks interpretability, it remains a model with outstanding precision. However, it is crucial to keep in mind the fact that this model is a so-called "black box", which is why explanatory algorithms like SHAP and LIME could be applied to it.

Now we may refer to the pseudocode provided in an article[Bui21] to grasp the essence of the random forest algorithm.

Algorithm 2 Pseudocode for the random forest (for both regression and classification cases)

input : M number of regression or classification trees
input : *type* can be either $classification$ or $regression$
for $i \leftarrow 1$ to M **do**
    Randomly sample the training data D with replacement to produce D_i
    Create a root node, N_i containing D_i
    Call BuildTree($N_i, type$)
end for
procedure BuildTree($N, type$)
    if *type* is $classification$ and N contains instances of only one class **then**
        return
    else
        Randomly select x% of the possible splitting features in N
        Select the feature F with the highest information gain to split on
        Create f child nodes of $N_1, ..., N_f$, where F has f possible values $(F_1, ..., F_f)$
        for $i \leftarrow 1$ to f **do**
            Set the contents of N_i to D_i, where D_i is all instances in N that match F_i
            Call BuildTree(N_i)
        end for
    end if
end procedure

4.3.2 Random forest classifier. Breast cancer data.

For the purposes of this section, the analysis is relied on a copy of the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset. Here was used **RandomForestClassifier** in **Sklearn**, one of the most popular implementation of random forest in Python. The selected hyper-parameters of a model are: n estimators (represents the number of trees in the forest) equals 1000, max depth (the number of splits that each decision tree in a model is allowed) is 100, the same as for the decision tree model (Appendix 9). Splitting the data into train and test sets was performed using **train_test_split**, where the test size equals 20 percent and random state is assigned to 42. The random state (or the random seed) is used to make sampling stationary, so we could reproduce the same result in the future.

Now we can move to code snippets. Let's start with required libraries (Appendix 5). Next, it is crucial to present the details of data acquisition, processing, and modeling itself. After that it is also important to check the key quality indicators such as the accuracy, precision, recall, and F1 scores:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{9}$$

In these formulas TP and TN stand for true positive and true negative, and FP and FN stand for false positive and false negative respectively. The accuracy score is one of the most popular benchmarks, and it shows the overall correctness of a model. "It is a measure of the reproducibility of results rather than their correctness"[Bri]. At the same time, precision measures how often a model is correct when predicting the target class (in our case it is "1", a patient who has the benign tumor). The recall score, also known as the sensitivity, demonstrates if a classifier can detect all objects of the target (benign) class. "The F-score describes an algorithm's performance on a scale of 0 to 1. An F-score of 1 indicates a perfect algorithm, and an F-score of 0 indicates an algorithm that has failed completely in either recall, precision, or both."[Akr] To sum up, this measurement is the aggregated one, which is really helpful when data is unbalanced. Determining if the dataset's classes are balanced can be challenging. Thus, the current dataset contains 212 malignant and 357 benign cases. It is essential to keep in mind such a distinction, particularly given the biomedical context of the application.

In addition, in cancer data modeling, the **recall** is a better measure than precision or accuracy. This score includes the FN predictions, so it is more important to minimize this type of misclassification than the FP ones. By using this approach, a medical researcher should detect as many patients with malignant cases as possible. Although some FP predictions may occur, those diagnoses can be corrected after additional checking. Unlike those genuinely battling cancer, who were mistakenly labeled as supposedly healthy. So, we will pay attention to this ratio in the further steps.

```
# Getting data
data = load_breast_cancer()
# Date processing stage
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# Model initializing
rf_classifier = RandomForestClassifier(n_estimators=1000, max_depth=100)
rf_classifier.fit(X_train, y_train)
random_forest_preds_classifier = rf_classifier.predict(X_test)
# Model's quality
print('The accuracy of the Random Forest classifier is
    :\t',metrics.accuracy_score(random_forest_preds_classifier,y_test))
print('The precision of the Random Forest classifier is
    :\t',metrics.precision_score(random_forest_preds_classifier,y_test))
print('The recall of the Random Forest classifier is
    :\t',metrics.recall_score(random_forest_preds_classifier,y_test))
print('The F1 score of the Random Forest classifier is
    :\t',metrics.f1_score(random_forest_preds_classifier,y_test))
```

```
The accuracy of the Random Forest classifier is :  0.9649122807017544
The precision of the Random Forest classifier is :  0.9859154929577465
```

```
The recall of the Random Forest classifier is :  0.958904109589041
The F1 score of the Random Forest classifier is :  0.9722222222222222
```

The selected benchmarks show that the classifier is quite precise on the test set. However, the recall ratio is not equal to 100%. It indicates that the model misclassifies a small percentage of patients with the malignant tumor, predicting the benign class for them. After all, this classifier can be helpful and we can keep it for the further analysis, which implies the use of posteriori black box interpretations and a comparison with a simple decision tree. In this case, the use of a more complex mechanism is explained by the fact that the decision tree handles less effectively with the forecast (see Appendix 9). All accuracy indicators are lower for the single decision tree classifier.

4.3.3 Random forest regression. California housing data.

The second example involves forecasting a continuous dependent variable. Now we should predict the median house values in Californian districts. For this purpose I used **RandomForestRegressor** in **Sklearn** as well. The selected hyper-parameters of a random forest model are: n estimators equals 1000, max depth is 100, the same as for the decision tree regression (see Appendix 10) and previous random forest models. The random state is also 42 to keep the datasets of the same content for the decision tree.

Certainly, assessing the quality of a regression is essential. To perform it, various indicators are available that can be employed for continuous target data explicitly. The most popular ones are mean squared error (MSE), root-mean-square error (RMSE), mean absolute percentage error (MAPE), and mean absolute error (MAE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{10}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{11}$$

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{max(\epsilon, y_i)}\right| \tag{12}$$

"where ϵ is an arbitrary small yet strictly positive number to avoid undefined results when y is zero."[Scia]

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{13}$$

These indicators suit well for both linear and non-linear models such random forests. It is worth noting that the widely used benchmark R^2 cannot be applied to non-linear models due to its underlying assumption[Neu10]. In addition, "for nonlinear regression, MSE is not an unbiased estimator of σ^2, but the bias is small when the sample size is large."[Was96] The current dataset contains 20640 observations initially. Thus, we may apply such indicators as MSE in this case.

```
# Getting data
california = fetch_california_housing()
# Date processing stage
california_X = pd.DataFrame(california['data'])
california_X.columns = california.feature_names
X_train, X_test, y_train, y_test = train_test_split(california_X,
                                                    california['target'], random_state=42)
# Model initializing
rf_regr = RandomForestRegressor(n_estimators=1000, max_depth=10)
rf_regr.fit(X_train, y_train)
random_forest_preds_regression = rf_regr.predict(X_test)
# Model's quality
```

```
print('The MSE of the Random Forest resgression is
    :\t',metrics.mean_squared_error(random_forest_preds_regression,y_test))
print('The RMSE of the Random Forest resgression is
    :\t',(metrics.mean_squared_error(random_forest_preds_regression,y_test))**0.5)
print('The MAPE of the Random Forest resgression is
    :\t',metrics.mean_absolute_percentage_error(random_forest_preds_regression,y_test))
print('The MAE of the Random Forest resgression is
    :\t',metrics.mean_absolute_error(random_forest_preds_regression,y_test))
```

```
The MSE of the Random Forest resgression is :  0.2907920913190029
The RMSE of the Random Forest resgression is :  0.539251417540096
The MAPE of the Random Forest resgression is :  0.1853312134351713
The MAE of the Random Forest resgression is :  0.36411437124448864
```

For now, drawing definitive conclusions about the model quality and precision is challenging. At least, we should compare these results with those of the decision tree. As it can be seen in the graph 2, the real median house values in Californian districts are positively correlated with the predicted ones. Higher real value corresponds to increased variability in house prices.

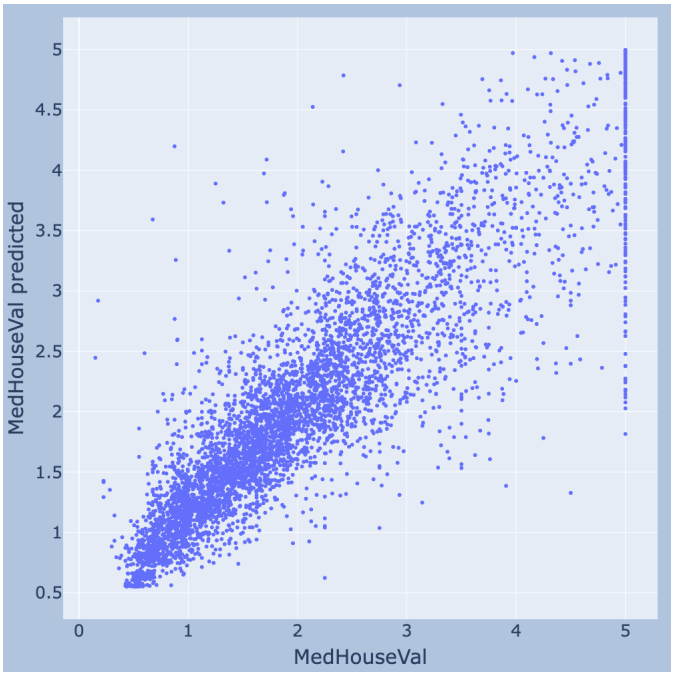

Figure 2: Median house values in Californian districts and the forecast.

As in the previous case, the use of a more complex mechanism is explained by the fact that the decision tree copes worse with the forecast (see Appendix 10). According to model quality metrics, the single decision tree regression show less precise results than the random forest model.

4.4 Imagery data. Convolutional Neural Networks.

A neural network is an ML model inspired by the principles of the human brain's interconnected nervous system. It is composed of interconnected nodes, or so-called neurons, organized into layers. Its number may vary. Neural networks are commonly used for classification, regression; image and pattern recognition; prediction of time series, and so forth.

In terms of the mathematical approach, the theoretical basis for NNs is not yet completely elaborated. It means that we cannot be ensured that deep models are capable of accurately predicting the response variable. It has been proven that an artificial neural networks with a single hidden layer can approximate any continuous function of multiple variables with any level of accuracy[Hay09]. Universal approximation theorem comes to this conclusion. Notwithstanding, it is quite obvious that analysts in most areas prefer to create deeper models because they demonstrate way better results[Yu18]. What is also important, researchers showed that the representation depth is beneficial for the image classification accuracy[Zis14].

4.4.1 Method description

Convolutional Neural Network is a type of deep learning model used mainly for image and video analysis. CNNs are a class of artificial neural networks designed to learn patterns and features from visual imagery. This type of NNs was initially suggested in 1989[Jac89]. "The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."[Cou16] CNNs consist of various layers, and most of them are specific to this type of neural networks. In the most vanilla case, CNNs contain convolutional, pooling, and fully-connected layers.

Let's start with the convolution itself. "In its most general form, convolution is an operation on two functions of a real-valued argument." [Cou16] Here is the standard definition for two functions f, g [Mat10]

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(u)g(t-u)\,dt \tag{14}$$

for f, g: $(-\infty, \infty) \to \mathbb{R}$, u is a dummy-variable

This method is applicable for the discrete convolution [Sla15]:

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k] \tag{15}$$

for all integer signals f, g defined on $\mathbb{Z}$
We need to note that the operation of convolution is commutative.

Now we can move to the definition of a convolutional layer. "A convolutional layer consists of a set of learnable filters. Every filter is spatially small in width and height, but extends through the full depth of the input volume."[Pér19]

Let's create some matrix for the discrete 2D convolution. One can find the example of calculations for the real photo 3.

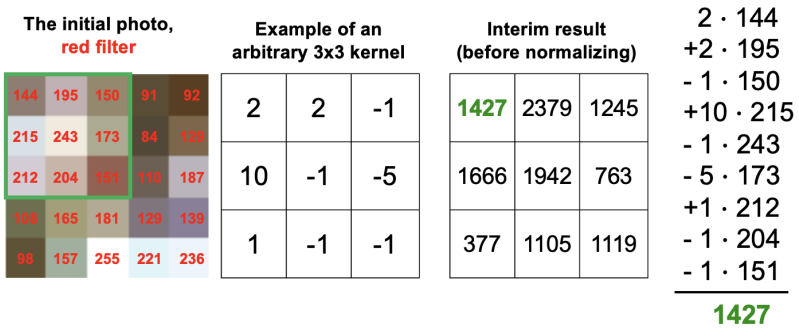

Figure 3: Example of convolution for the real image from the trains set.

Though it shows the only filter (the red one), other filters (the green and blue ones) undergo the same process. As can be noticed, the final results are out of the interval $[0, 255]$, which is a standard for RGB images. So, we should provide normalization to transfer these values into the suitable format 16. Here is how (the pixelwise process):

$$x^{'} = \frac{x - x_{min}}{x_{max} - x_{min}} * 255 \tag{16}$$

$$\frac{1427 - (-806)}{2379 - (-806)} * 255 \approx 178.78022 \tag{17}$$

The non-integer result can be rounded, for example, by eliminating the decimal part, so we can use 178 instead of 178.78022 17 for simplicity. The final result of an arbitrary selected 3×3 kernel is provided below 4, the image is normalized.

Figure 4: Convoluted image. The key features (letters, sign shape) became more distinct.

Pooling layer is another fundamental type of the CNN structure. "The function of a pooling layer, also known as a subsampling layer, is to progressively reduce the spatial size of the representation, to reduce the amount of parameters and computation in the network. The pooling is typically done using the average or maximum function, applied to the subset in consideration."[Pér19]. The max-pooling of a part of the convoluted image is provided below 5. There was found that max-pooling might lead to faster convergence, select superior invariant features, and improve the model generalization. [Beh10]

Figure 5: Max-pooling of the convoluted image.

Fully connected layers connect each neuron in one layer to every neuron in the adjacent layer. They contain learned weights and biases, apply non-linear activation functions, and are located at the end of networks for making final predictions [Pér19].

Another layer type we will look at is a dropout layer. It is a quasi-regularization technique used in neural networks, particularly in deep learning models, to prevent overfitting. This issue occurs if a model performs very well on the training data but fails to generalize effectively to test (new) dataset. Dropout minimizes this risk while deactivating a cluster of neurons during each training iteration. This technique was initially suggested in 2014 [Sal14]. In this work this method is used to prevent overfitting. The researchers came to the solution, that the dropout is similar to the regularization technique.

4.4.2 Implementation of Convolutional Neural Networks

Let's delve into the practical aspect and look into the Python code. First of all, it is essential to describe the architecture of the selected CNN model. I decided to use an already existing framework, which showed high accuracy on the test set. There are many almost identical implementations (e.g. [Gup23], [Cos], [Sac21]), and I used [Mov] as a main source. However, I also included some modifications too, especially those which enable users to control over training (early stopping under some condition) and save the most precise model on the validation set. In the figure6 one can notice that this particular model contains three convolution layers, two max-pooling layers, one flatten one, two dense layers, and three dropouts. Initial picture is of 50 $\times$ 50 size in pixels, so that can be too detailed and sophisticated for efficient and relatively fast model training. Thus, one should include max-pooling for reducing the feature map. Which is also important, a model should omit unimportant, irrelevant, and random artifacts in the pictures. However, it has to detect crucial patterns, which help to classify traffic signs correctly. To accelerate this process, convolution layers are being used, as well as dropout layers, that minimize the risks of overfitting.

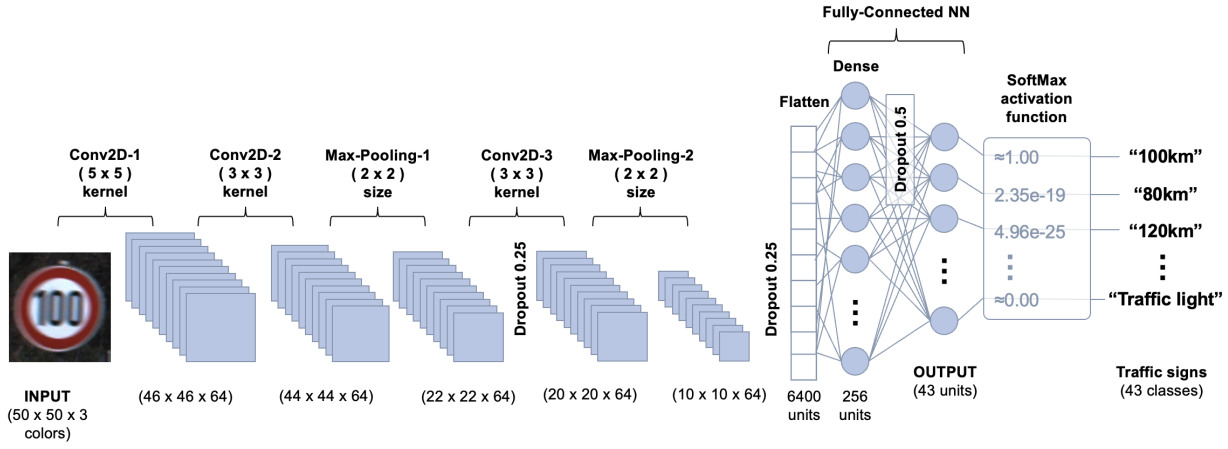

Figure 6: The framework of the CNN. Initial framework: [Mov]; visualization is based on: [Rat21].

For the interim layers, the ReLU activation function was used:

$$f(x) = max(0, x) \tag{18}$$

It suits well because ReLU helps mitigate the vanishing problem of gradients, introduces non-linearity to the model while remaining computational efficient and providing fast convergence[Cai18].

Meanwhile, employing ReLU is not suitable for the final layer of a CNN classifier. There activation functions based of probability distribution must be considered instead. So, Softmax19 is a good choice due to the possibility to work with multiclass classification and the training objective (cross-entropy loss in our case)[Zhu21]:

$$\sigma(\mathbf{z})_i = \frac{e^{\mathbf{w}_i^\intercal \mathbf{z}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j^\intercal \mathbf{z}}}, \text{ for } i = 1, ..., K \tag{19}$$

where $\mathbf{z} = \psi(\mathbf{x}) \in \mathbb{R}^H$, input $\mathbf{x} \in \mathbb{R}^D$, final-layer weight $\mathbf{W} \in \mathbb{R}^{H \times K}; \sigma : \mathbb{R}^K \to [0,1]^K, K \geq 1$ - target classes

Resuming the scheme6 discussion, it is essential to highlight that the last layer consists of predictions. The initial photo was drawn randomly from the test set. There are calculated probabilities for each class from the whole list of traffic signs. These probabilities are arranged in descending order starting with the most probable class. Here the prediction is correct, this sign stands for the speed limit of 100 km/h. The least probable label is the traffic light, and its value is extremely close to 0 for the format `np.float64`, so it is rounded to 0. As can be noticed, probabilities of other incorrect classes are also extremely close to 0.

Now we can move to the Python code itself, see Appendix 6. Necessary libraries and functions are available below to carry out all the necessary procedures for achieving a well-trained CNN model. The key package selected for building neural networks was Keras, as one of the best libraries for image recognition. So, "Keras is Python's high-level deep learning platform that can operate on top of TensorFlow."[Sub20]

The second step is creating of the framework for the subsequent analysis. The relevant code snippet can be found in Appendix 7. This stage is based on the tutorial[Mov], however it also modified and partially simplified. In this step a researcher should also think of getting data from folders. The data structure here includes two main folders named "Train" (39209 photos) and "Test" (12630 photos). Then, the first one is being split into 43 classes. The test folder contains a simple list of images, so the data is not labelled. After this procedure data collection needs to be implemented. As shown below, it can be done via nested For loop.

```
# based on
    https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
for i in range(classes) :
    path = "my_path/Train/{0}/".format(i)
    Class = os.listdir(path)
    for a in Class:
        image = cv.imread(path+a)
        imageRGB = cv.cvtColor(image, cv.COLOR_BGR2RGB)
        image_from_array = Image.fromarray(imageRGB)
        size_image = image_from_array.resize((height, width))
        data.append(np.array(size_image))
        labels.append(i)
Cells = np.array(data)
labels = np.array(labels)
# Randomize the order of the input images
s = np.arange(Cells.shape[0])
np.random.seed(43)
np.random.shuffle(s)
Cells = Cells[s]
labels = labels[s]
```

Next, a researcher should split data into train and validation sets. Of course, there are some other advanced resampling techniques such as the cross-validation or bootstrapping. Nonetheless, taking into account the size of the train dataset, it is sufficient to limit the task to a simple division into the train, validation, and test sets. Thus, the final sets are the 80%-train (31368 images),

20%-validation (7841 images), and test (12630 images). The last one is excluded during classifier training as we aim to assess the model accuracy on completely new data.

```
# code snippet from
    https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
# Spliting the images into train and validation sets
(X_train,X_val)=Cells[(int)(0.2*len(labels)):],Cells[:(int)(0.2*len(labels))] # 80%
    train (31368 images), 20% validation (7841 images)
X_train = X_train.astype('float32')/255
X_val = X_val.astype('float32')/255
(y_train,y_val)=labels[(int)(0.2*len(labels)):],labels[:(int)(0.2*len(labels))]
y_train = to_categorical(y_train, 43)
y_val = to_categorical(y_val, 43)
```

Following this step, we can proceed to initialize the CNN itself. The main specifics were already discussed, and the model architecture is visualized6. Adam[KB17] was chosen as the optimizer since this method has proven to be effective in many applications of machine learning. Some other variations such as SGD and RMSprop were checked too, but the accuracy on the test set for those classifiers was slightly lower. It is worth noting that some researchers overlook the importance of early stopping and saving the best model. Early stopping, as a form of regularization, prevents overfitting and the problem of unexpected large execution time. However, there is also "tradeoff between training time and generalization error"[Pre02], so one needs to calibrate the criterion of early stopping to get better results. Afterwards, it is essential to save the best possible CNN classifier, although in most tutorials the last model (not necessarily the best one) is being used in further analysis. To perform this, we need to update the code and include both Keras' functions `EarlyStopping` and `ModelCheckpoint`, so users should save the best model. In this case, it denotes the model with the highest level of `val_accuracy`. The final version of model architecture, hyperparameters, and other settings is provided below.

```
# based on
    https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
model = Sequential()
x = Conv2D(filters=64,kernel_size=(5,5),input_shape=X_train.shape[1:])
model.add(x)
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
model.compile(
    loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['accuracy']
```

```
)
earlyStopping = EarlyStopping(monitor='val_accuracy', patience=10, verbose=1,
    mode='max')
mcp_save = ModelCheckpoint('best_model.mdl_wts.hdf5', save_best_only=True,
    monitor='val_accuracy', mode='max')
epochs = 100
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
    validation_data=(X_val, y_val), callbacks=[earlyStopping, mcp_save])
```

The model training process is visualized in the figure7. There were 23 epochs and, as it was already stated, the best model in terms of validation accuracy, the thirteenth epoch, was saved. The training process is efficient, which is also evident in the plot. The accuracy for both the train and validation sets increases while the loss gradually reduces over the epochs. One can notice an unusual behavior when the validation accuracy is higher than the training accuracy and the validation loss is less than the training loss respectively. This situation is not rare, and Keras FAQ says[Ker]: "A Keras model has two modes: training and testing. Regularization mechanisms, such as Dropout and L1/L2 weight regularization, are turned off at testing time." Here I would like to mention that I used dropout layers. "They are reflected in the training time loss but not in the test time loss. Besides, the training loss that Keras displays is the average of the losses for each batch of training data, over the current epoch. Because your model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. This can bring the epoch-wise average down. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss." Thus, it is now that unexpected as it might seem.

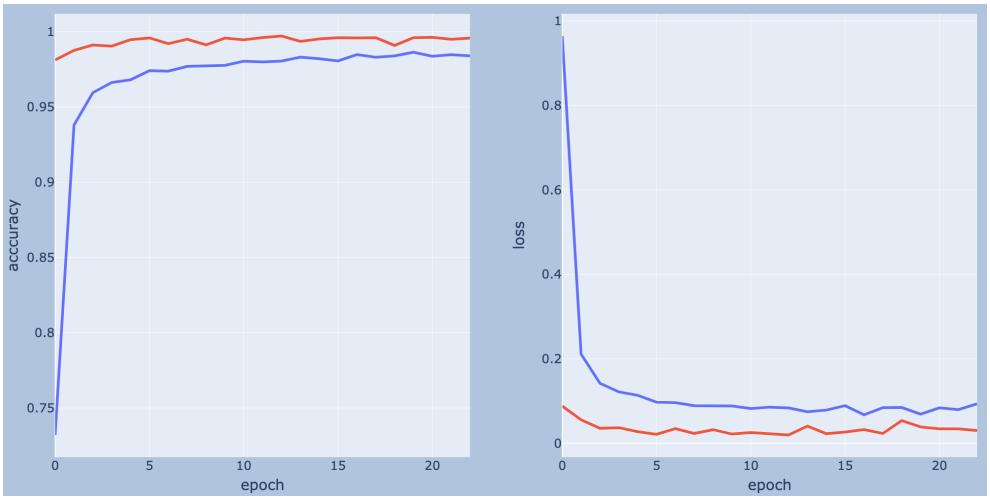

Figure 7: Accuracy and loss of the final CNN classifier. Blue line: training data, red line: validation data. Possible cause of the discrepancy: dropout mechanism (see explanations above).

However, relying solely on the benchmarks for the training and validation sets is way to be enough to assess the model quality. This implies that we also need to try this classifier using completely new data. One can find the code snippet below, so the final model demonstrates exceptional accuracy ($> 0.97\%$) on the test data.

```
# based on
    https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
# Predicting with the test data
y_test = pd.read_csv("my_path/Test.csv")
labels = y_test['Path'].to_numpy()
y_test = y_test['ClassId'].values
data=[]
for f in labels:
    image = cv.imread('my_path/Test/'+f.replace('Test/', ''))
    imageRGB = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    image_from_array = Image.fromarray(imageRGB)
    resized_image = image_from_array.resize((height, width))
    data.append(np.array(resized_image))
X_test=np.array(data)
X_test = X_test.astype('float32')/255
predict_x = model.predict(X_test)
classes_x = np.argmax(predict_x,axis=1)
print("The accuracy of the CNN classifier on the test set is: ",
    accuracy_score(classes_x, y_test))
```

```
The accuracy of the CNN classifier on the test set is:  0.9733174980205859
```

5 Black box explainers

5.1 LIME

LIME (Local Interpretable Model-Agnostic Explanations) is an explainer, which purpose is to provide interpretability for induvidual predictions made by deep learning models such as neural networks or random forests. LIME focuses on explaining the local behavior of a model in relation to a particular instance, but not the model as a whole. There is a local (linear) approximation of the model decision boundary.

LIME enables us to comprehend the local behavior of black box models. This method requires approximating the ML's decisions with interpretable models within a specified neighborhood. LIME points out important features and provides insight into how changes to those features might impact the model predictions. This algorithm provides case-level interpretability of the model and helps build trust and understanding of black box models.

Suppose we have a black box regression or classification model $f : \mathbb{R}^p \to \mathbb{R}$ which prediction for instance x we want to explain using a model $g : \mathbb{R}^p \to \mathbb{R}$. The model g can be any self-explainable statistical model, $g \in G$. Thus, the set G is a class of potentially interpretable models such as LASSO, LARS, ridge regression, elastic net, or decision trees e.g. LIME derives feature importance in prediction for instance x by minimizing the loss function L under the low model complexity $\Omega(g)$ of the interpretable model g. An original generalized formula summarizing LIME for local surrogate models with interpretability constraint $\Omega(g)$ was suggested in the literature: [Gue16], [Mol22]:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \Big\{ L(f, g, \pi_x) + \Omega(g) \Big\} \tag{20}$$

Here we have the vector of "explanations" ξ, which contains coefficients for p variables from the model g, for the input instance x, which is a data row from the original dataset (can be also an image).

To proceed with optimisation explanation, we need to define the sample used because this method takes only one observation x as an input. Generation of the neighborhood sample X for the chosen instance x is the first step in the LIME algorithm. Transformation of images into matrices is explained in the further part of this section. Observation in the neighborhood sample $x^{'} \in X$ is generated as $x^{'} \in \{0, 1\}^p$ from the original instance x. This step includes changing the color of superpixels while working with images. Therefore, we have instance x, the original image, and N images which differ from x and each other by a combination of superpixels which colors are changed (e.g. set to black when a superpixel is assigned a zero value).

Now we can return to the equation 20 and discuss the mechanism how feature importance is derived in LIME. Let's look into the first part of this minimization problem, $L(f, g, \pi_x)$. For the chosen instance x, N permutations are generated forming a neighborhood sample as described above. The weighted loss function L (weighted MSE, MAE, etc.) shows the precision of the explainer's g results in comparison to the predictions of the original ML model f (in our case that is CNNs and random forest) for the neighborhood sample X. In short, we obtain predictions for the neighborhood sample from the black box model f and use them to train the interpretable model g. During this step it is important to calculate the similarity between the instance x and

each perturbation $x^{'}$. Kernel π_x with some constant width σ is the proximity measure which assigns higher weights to perturbations which are more similar to the original instance x as it takes the distance between them. Usually, researchers use the L2 distance (in this context it is interchangeable to the Euclidean distance) in case of cross-sectional data and images[Gue16]:

$$D(x, x^{'}) = ||x - x^{'}||_2 \tag{21}$$

$$\pi_x(x) = e^{\frac{-D(x,x^{'})^2}{\sigma^2}} \tag{22}$$

$L(f, g, \pi_x)$ is a measure of how unfaithful g is in approximating f in the local area. Therefore, we have the formula of the weighted sum of squared distances (errors) between predictions of the two models, the complex f and simplified g, and the loss function L can be rewritten as:

$$L(f, g, \pi_x) = \sum_{x,x^{'} \in X} \pi_x(x^{'}) \Big(f(x^{'}) - g(x^{'}) \Big)^2 \tag{23}$$

$$= \sum_{x,x^{'} \in X} e^{\frac{-D(x,x^{'})^2}{\sigma^2}} \Big(f(x^{'}) - g(x^{'}) \Big)^2 \tag{24}$$

This is the weighted loss function which is minimised in model g. In this paper, ridge regression is used as a model g to obtain feature importance for each tabular feature or superpixel. In other words, feature importance for the used random forest regression model is calculated using the coefficients from the ridge regression. Analogically, importance of superpixels is determined from the predictions of ridge regression for each class.

If the black box model f solves a classification problem, as in this paper for the case of traffic signs, probabilities for each of the classes are predicted by the model g. For example, if there are 5 classes, then we run the model g 5 times on the neighborhood sample to explain the probability of the observations x' to belong to each class. If the black box model f solves a regression problem, a single vector is to be calculated. The resulting feature importances taken as those corresponding to the highest probability for classification from model f.

The second part in the equation 20 amplifies that LIME assumes choosing the model g as simple and interpretable as possible, in contrast to the black box model f, which can be described as the low model complexity $\Omega(g)$. Usually it means fewer exogenous factors in practice, but an interpretable model can be already sufficient.

Overall, we need to minimize $L(f, g, \pi_x)$ while having $\Omega(g)$ be low enough to be interpretable by humans. Thus, the first part of this minimization denotes the need to get the precise model, while the other one stands for keeping the resulting model relatively simple.

Now we can rewrite equation 20 for the ridge regression, since it will be the selected regularization option:

$$\xi(x) = \hat{\beta}^{Ridge} = \underset{\beta}{\operatorname{argmin}} \Big\{ \sum_{i=1}^{N} e^{\frac{-D(x,x_i^{'})^2}{\sigma^2}} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}^{'} \beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \Big\} \tag{25}$$

Equation 25 shows how coefficients are determined for a vector of continuous values in case of regression or probabilities for each possible class in case of multiclass classification. Here y_i denotes $f(x^{'})$, which is the output vector of the complex model for an n-th instance $x^{'}$ belonging to the neighborhood sample X. N equals to the number of perturbations (chosen by a person), $x^{'} \in \mathbb{R}^{N \times P}$ is a dataset that consists of N perturbations and P variables. Weights are determined according to equations 22 and 21.

To sum up the algorithms below, LIME can be described as follows. A researcher determines an instance of prediction to be explained and the number of perturbations first. Then, the algorithm itself starts with neighborhood generation. Here we need to create a new dataset based on an initial one. This procedure is performed by introducing some relatively small perturbations into the original training data while keeping the target variable fixed. The next stage is weighting. LIME initiates weights to the neighborhood instances, which are determined from their "similarity" to the original instance, here one should use L2 distance. Those examples that are closer in feature space to the initial instance are assigned with a higher weight. Then we need to obtain predictions from the underlying black box model for each item in a set of perturbed data. After that, to provide explanations, LIME builds an interpretable or explainable model that approximates the behavior of the black box model within the local neighborhood. This interpretable model should be simple and understandable such as linear regression, decision tree, LASSO, etc. Ridge regression was chosen in this thesis, since this model copes with multicollinearity and does not force any coefficients to zero. So, we need to assess the feature importance according to the trained interpretable model. Here researchers use the model coefficients to evaluate the factor contribution for a particular instance (an observation, an image). Thus, the feature importance demonstrates us how much each feature influences the results made by the black box model for a particular forecast. Naturally, the impact of a regressor can be positive, negative, or nearly negligible. Finally, one should keep in mind the fact that these explanations linearly approximate the influence of most important factors for an individual instance. However, to get more general and comprehensive understanding of the model behavior, researchers may repeat the procedure for other instances too.

Algorithm 3 Pseudocode for LIME for tabular data

input : black box machine learning model f, interpretable model g, number of perturbations N
input : instance x (a data row), number of features P
input : similarity kernel π_x (e.g. exponential smoothing), Neighborhood sample X around x
output : predictions of the black box model for the neighborhood sample y
output : feature importance for f-model prediction for the instance x derived from model g
$X \leftarrow \{\}$
$y \leftarrow \{\}$
for $i \leftarrow 1$ to N **do**
    $x_i^{'} \leftarrow neighborhood\_data(x)$ (perturbed row)
    $X[i,] \leftarrow x_i$ (save perturbation to the matrix for them)
    $D \leftarrow$ L2_distance$(x, x_i^{'})$
    $W[i] \leftarrow \pi_x(D)$
    $y[i] \leftarrow f(x_i^{'})$ (obtain prediction from model f)
end for
$\hat{\beta}_x^w \leftarrow RidgeRegression(X, y, W)$

Algorithm 4 Pseudocode for LIME for imagery data

input : black box machine learning model f, interpretable model g, number of perturbations N
input : instance x ($a \times b \times$ 3 colors), where a and b are the image's length and height in pixels
input : number of superpixels P, similarity kernel π_x (e.g. exponential smoothing)
input : matrix to define superpixels $\mathrm{M}^{a \times b}$ contains values from 1 to P
output : neighborhood sample X around instance x
output : Predictions of the black box model for the neighborhood sample y
output : Feature importance for f-model prediction for the instance x derived from model g
$X \leftarrow \{\}$
$y \leftarrow \{\}$
for $i \leftarrow 1$ to N **do**
    $x_i^{'} \leftarrow x$ (copy instance x)
    $superpixels \leftarrow rand(0, 1, P)$ (generate a list (length of P) consisting of 0 and 1)
    $mask \leftarrow BooleanMatrix \in \mathbb{R}^{a \times b}$
    for $k \leftarrow 1$ to a **do**
        for $q \leftarrow 1$ to b **do**
            $mask[k, q] \leftarrow False$ (fill mask with False for each superpixel)
        end for
    end for
    for $j \leftarrow 1$ to P **do**
        if superpixels[j]==1 **then**
            $mask[M == j] \leftarrow True$ (replace values with True for pixels in a j-th superpixel)
        end if
    end for
    $x_i^{'}[mask] \leftarrow 0$ (turn off superpixels - fill with black color)
    $X[i,] \leftarrow x_i$ (save perturbation to the matrix for them)
    $D \leftarrow$ L2_distance$(x, x_i^{'})$
    $W[i] \leftarrow \pi_x(D)$
    $y[i] \leftarrow f(x_i^{'})$ (obtain prediction from model f)
end for
$\hat{\beta}_x^w \leftarrow RidgeRegression(X, y, W)$

5.1.1 Implementation of LIME for tabular data

Explanations for classification. Here we refer to the outcomes of the random forest models from the previous part. So, the results of the random forest classifier based on breast cancer data are provided below8 9. There were randomly selected data rows from the test set, which was correctly classified as "0", malignant tumor and another one, class "1", the benign case. The same test instances will be used for SHAP in the subsequent section.

In the graph8 for the first patient the worst concave point, worst area, worst perimeter, etc. are higher than those for an average patient. There are no effects, which could enlarge the probability of having the benign prediction. Thus, the probability of the malignant cancer here is assessed as 1.00 (the difference between this value and 1 is negligible). The class for this patient was predicted correctly.

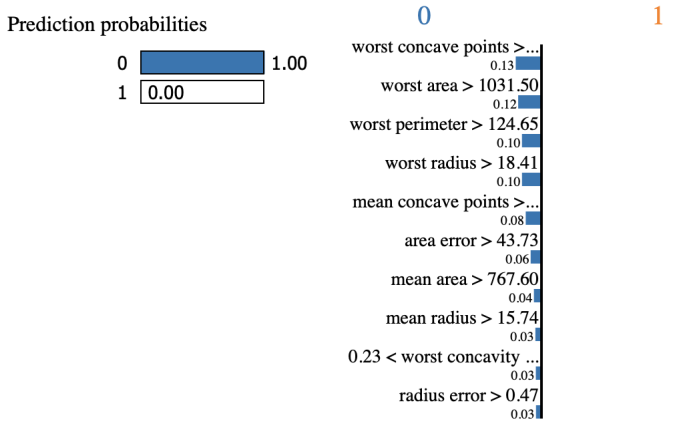

Figure 8: LIME explanations for the classifier based on breast cancer dataset. Malignant case.

Another example was correctly classified as the benign case9. The probability of having "1" equals 0.97. Worst area, worst perimeter, and the worst radius are factors that increased probability of the benign prediction. All of these values are less that the average ones.

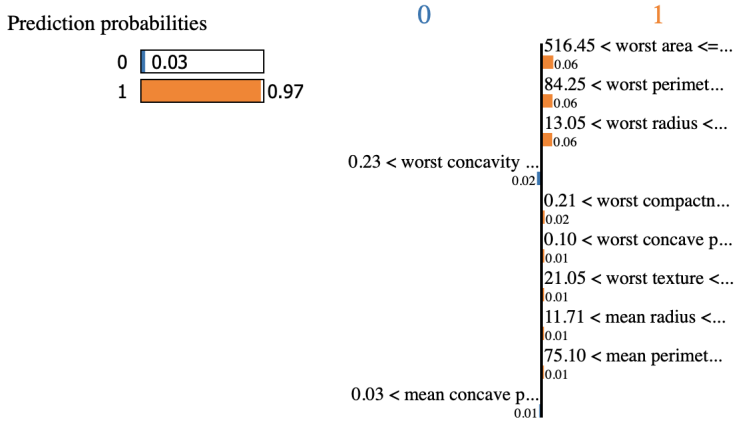

Figure 9: LIME explanations for the classifier based on breast cancer dataset. Benign case.

I included some comparison of these results and those made by a simple decision tree classifier. In Appendix 9 one can find that, according to a simpler model, the most determinant factors are the mean concave points, worst radius, and worst concave points. These conclusions are quite similar to the results of LIME for the random forest classifier. The discrepancies in the order of importance of factors are conditioned by the fact that LIME represents local explanations, and decision trees are generalized for the entire dataset.

Explanations for regression. Now one can notice the explanations for the California housing dataset10. Here we draw the random example from the test set too. The same data row will be used for the SHAP explanations. So, the median income in block group (MedInc) is higher than the overall average value, and that increases the predicted house price. However, the average number of household members (AveOccup) is too high, so it decreases the predicted value.

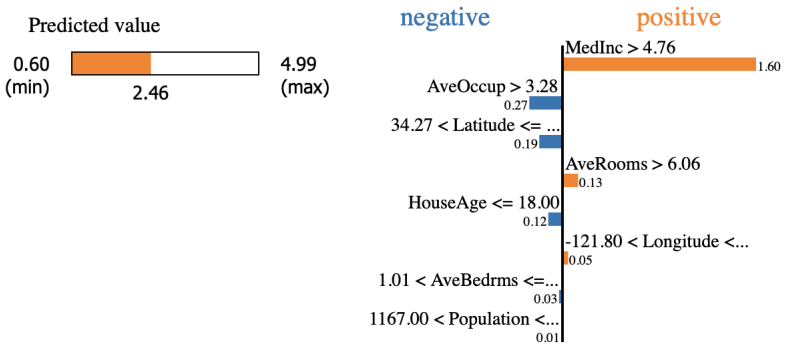

Figure 10: LIME explanations for the test instance of the RF regression based on California housing dataset. The real house price equals 2.186.

According to Appendix 10, the decision tree regression the median income in block group (MedInc) is also the most important factor. The second one is the average number of household members (AveOccup). The direction of change of the resulting variable is the same as in case of LIME explanations for the random forest regression.

5.1.2 Implementation of LIME for imagery data

Now we can to start with some real image. I selected the traffic sign of the 30 km speed limit. This picture is 50×50 pixels in size and consists of a combination of 3 colors - red, green, and blue. All the necessary packages and functions are presented in Appendix 8. Please note that there is no explicit importing of the LIME library for Python, though it might be done via the use of `import lime`. It is the default version of LIME. Instead, I took the initial code from[LIM], sliced it into several parts that I was interested in so I could get the interim results such as 14 or 15.

Figure 11: Selected picture to be classified with CNNs and explained with LIME and SHAP.

For the first step, we need to use this picture as a test. So, there are results 12 that show probabilities for each class. The likelihood of having "30km" is 1.0 according to the Python's output.

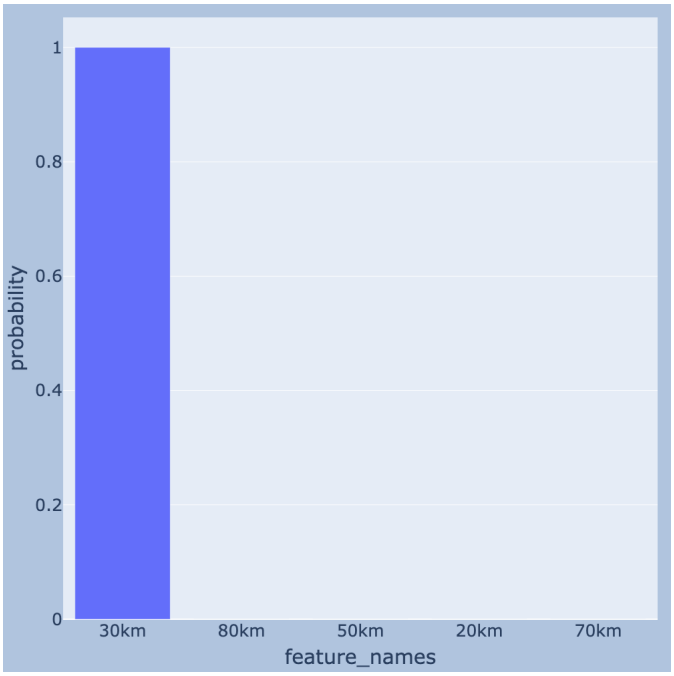

Figure 12: Top-5 predictions for the sign 30km speed limit. Predicted probabilities for the wrong classes are close to 0.

After that, we create a neighborhood sample, or set of some perturbed images, which are similar to the initial picture, but not the same. Here we consider 20 perturbations. There is the example14.

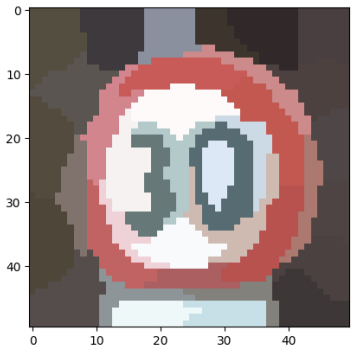

Figure 13: Segmented image with 100 superpixels to be used in the further analysis.

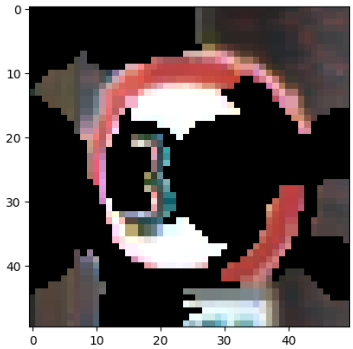

Figure 14: Randomly selected picture, which is similar to the initial one 11.

Visualization of randomly selected superpixels for theese 20 generated images, or instances, is provided below. The darker color depicts the excluded segments, while the lighter ones show the kept superpixels 15.

After that, we need to process them all. Firstly, we predict classes using the real black box model, which is CNN in our case. Here the outputs are the probabilities of each class: “30km”, “80km”, “50km”, “20km”, and “70km”. For the second, we calculate the distance between the real photo 11 and generated ones. Then, we use these distances to calculate the weights for the ridge regression. Here the probabilities of each class are calculated, the dataset looks like the matrix 15. Afterwards,

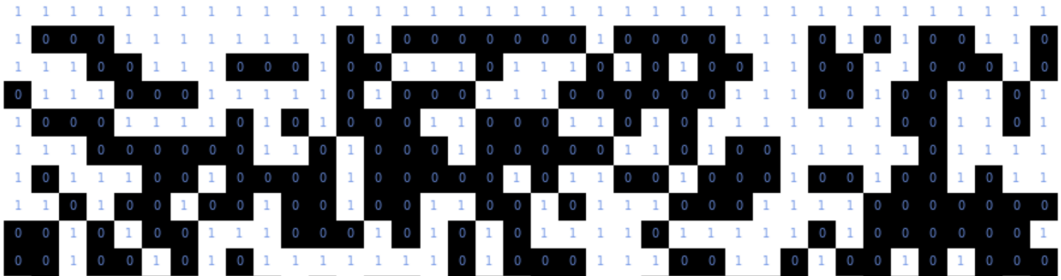

Figure 15: Matrix, which shows randomly kept (lighter color) and hidden (darker color) superpixels for the first 10 instances. Here columns indicate number of a superpixel, which can be either kept or hidden. The first row is the initial image without any hidden parts.

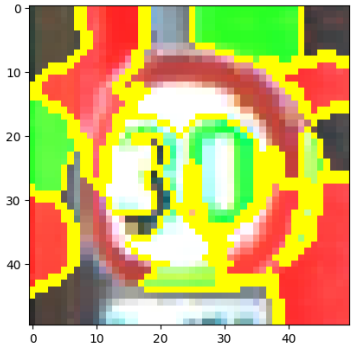

Figure 16: Top 10 LIME explanations by importance for the class “30km”, the correct label.

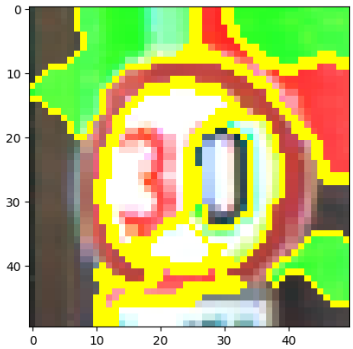

Figure 17: Top 10 LIME explanations by importance for the class “80km”, the wrong label.

we get the β_p for each segment. The impact of some superpixel p on the overall prediction can be positive, negative, and neutral ($\beta_p \approx 0$).

Then, we would like to test the explanations. So, let’s select the class with the highest probability, for our case both CNN and ridge regression predicted “30km”, which is the correct output. Here 12 we got the top 5 results for the most probable class, “30km”. The green segments depicts the most superpixels with the highest positive impact on the prediciton “30km”. There are no superpixels with the negative impact on this prediction.

However, we can also look into the second best prediction, which is “80km” 17. Here it can be noticed, that most impactful superpixels are colored with red, which means that their coefficients $\beta_1, ..., \beta_p$ are negative. There is the only segment, which increases the overall probability of “80km”, and that is just a part of a circle.

To sum up, LIME explanations for 2 most likely classes, “30km” and “80km”, were provided in this section. However, the probability of “30km” is higher, and that is correctly classified image. LIME shows that the parts of “30”, especially zero, increase the chance that it is “30km”, as well as some parts of a red circle containing the number 30. Nonetheless, some background is also used by the ridge regression. It means that the underlying black box model, CNN, is capable of taking background as a part of a traffic sign classification, which is a serious issue. This fact has to be considered in the further work with this neural network.

5.2 SHAP

SHAP (Shapley Additive Explanations) is a popular ML technique to interpret results of a black box model. This approach is based on the concept of the Game Theory and the Shapley values. These values are assigned to each factor in the forecast, indicating its contribution to the result of the forecast. The main idea behind SHAP is to understand the contribution of each feature by measuring how predictions change when different combinations (or so-called coalitions) of features are included or excluded. Generally, SHAP is more advanced and provable than LIME. This algorithm can be used in two ways: at the single instance level and for explanation of the feature contribution for the entire dataset, unlike LIME, which works locally only. At the same time, SHAP is more computationally complex and time consuming.

By examining the Shapley values, analysts can understand which features are driving the model predictions and to what extent. This information helps in interpreting the model behavior, identifying influential features, detecting biases, and building trust by providing transparent explanations for the model outcomes.

Importantly, calculation of Shapley values can be computationally expensive, especially for complex models and large feature spaces. Various approximation techniques and algorithms have been developed for efficient estimation of Shapley values, making SHAP applicable to a wide range of models and datasets. For instance, a random and limited set of coalitions can be used instead of the whole list of possible options.

Let's turn to the basic theory of what is the meaning of Shapley value[Sha53]. Since this substance originally appeared in game theory, it is determined based on the value function val assigned to players in the feature subset S. The Shapley value of a feature value is calculated by considering its contribution to the overall payout, which is weighted and aggregated across all possible combinations of feature values[Mol22]:

$$\phi_j(val) = \sum_{S \subseteq \{x_1,...,x_p\} \setminus \{xj\}} \frac{|S|!(p-|S|-1)!}{p!}(val(S \cup \{x_j\}) - val(S)) \tag{26}$$

Here $val_x(S)$ means the predicted value for feature values in set S, which is obtained by considering the marginalization of features that are not included in set S. Suppose we have a ML model $f : \mathbb{R}^p \to \mathbb{R}$ which prediction $\hat{f}(x)$ we want to examine. The vector x represents the feature values of the instance to be explained, while p denotes the total number of factors in a model. Here is the formula for $val_x(S)$ [Mol22]:

$$val_x(S) = \int \hat{f}(x_1, ..., x_p) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X)) \tag{27}$$

The procedure in more detail can be described in this way. Our focus lies in understanding the impact of each feature on the prediction of a specific data point. In the case of a linear model, determining the individual effects is straightforward. Let's examine the prediction of a linear model for a single data instance:

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + ... + \hat{\beta}_p x_p \tag{28}$$

Generally, the goal is to obtain the contribution ϕ_i of the i-th feature on the prediction $\hat{f}(x)$, which can be determined as:

$$\phi_i = \beta_i x_i - E(\beta_i X_i) = \beta_i x_i - \beta_i E(X_i) \tag{29}$$

Typically these weights are not provided in a ML model. Thus, we need to try another solution. In order to calculate feature's Shapley value, we need to consider possible combinations of all values for the feature set when the feature of interest is set to a random value and model predictions for them. For example, all possible combinations for the case of having 3 features is provided below. Here "features" can represent both cross-sectional variables in case of tabular data and dummy-variables for (super)-pixels in case of imagery data. "Random" cells means small perturbations of continuous data. Randomness stands for the random choice whether to include the (super)-pixel or not. Those cells that are named "real value" contain the initial values. For imagery case, it means that the (super)-pixels are kept untouched. Again, the complexity of SHAP increases exponentially with the growth of the feature number.

All possible coalitions for 3 features			
Coalition	Feature 1	Feature 2	Feature 3
π_1	random	random	random
π_2	real value	random	random
π_3	random	real value	random
π_4	random	random	real value
π_5	real value	real value	random
π_6	real value	random	real value
π_7	random	real value	real value
π_8	real value	real value	real value

Here I provide my version of the SHAP algorithm partially based on the one suggested in another article[Hus20].

Algorithm 5 Pseudocode for SHAP (applicable for both tabular and imagery data)

input : number of features ((super)-pixels) P, data matrix X, instance (data row) x,
set of features coalitions K, number of random coalition draws M
output : Shapley value for all features (vector $\phi(x)$)
Definitions : $\hat{\pi}_i$ is a coalition of features where x_i has real value and random values are
assigned to features x_j $\forall i \neq j$. π is a completely random coalition of features
for $i \leftarrow 1$ to P **do**
    for $j \leftarrow 1$ to M **do**
        Draw $\hat{\pi}_i^j$ and π_i^j randomly from K
        $\phi_i^j \leftarrow \hat{f}(\hat{\pi}_i^j) - \hat{f}(\pi_i^j)$, the marginal contribution calculation (28)
    end for
    $\phi_i(x) \leftarrow \frac{1}{M} \sum_{j=1}^{M} \phi_i^j$, we have the computed Shapley value as the average
end for
return $\phi(x)$

The last step was described in [Kon14], so it is an example of approximation with Monte-Carlo integration. To obtain all Shapley values, the process needs to be iterated for each individual

feature.

To sum up, SHAP considers a sufficient number of possible coalitions of features to obtain a consistent estimates and computes their respective Shapley values. It begins with an empty set of features and gradually adds one feature at a time to form different feature combinations. Each coalition contains some feature permutations, so that a real value n-th factor can be either kept or replaced with a randomized value. The last one, in the case of imagery data, denotes either 0 (the segment is being painted black) or 1 (the segment is being kept). However, in practice the use of the whole lists of coalitions is not the best solution due to exponentially increasing model complexity: the set size equals 2^n, where n is a number of parameters. Usually, analysts use random sampling and limit the maximal number of coalitions. Then, for each feature combination, SHAP compares the prediction made with the given combination to the prediction made, potentially, without the real value for the feature. The difference between these two predictions represents the contribution of the added feature. SHAP calculates the Shapley value by averaging the contributions across all possible feature combinations, taking into account their respective probabilities of occurrence. This assigns a value to each feature, representing its average contribution to the predictions. The Shapley values obtained through this process provide insights into the importance and impact of each feature on the predictions. Positive Shapley values indicate features that increase the prediction compared to the baseline, while negative values suggest features that decrease the prediction.

5.2.1 Implementation of SHAP for tabular data

In this paper the Python package SHAP is used for tabular and imagery datasets. Notably, the outcomes of SHAP are sparse, meaning that some Shapley values can be estimated as zero. This characteristic sets SHAP apart from the traditional Shapley values method. Also, in this part SHAP explanations will be interpreted locally and globally for the whole set of predicitons, since it is possible to provide for tabular data.

Explanations for classification. Now we are capable of implementing SHAP for the classification model based on the breast cancer dataset. We can start with some local explanation, we can take the first row from the test set18. Here the model predicts the class "1", which means **benign**, and the real value is the same. The probability of getting the benign class equals to 0.97. The results of SHAP show that the worst concave points size of 0.1015 (the overall dataset average is 0.1146), the mean concave points size of 0.03821 (the average is 0.0489), the worst perimeter size of 96.05 (the average is 107.26), and other factors colored with red enlarge the chance of having the benign tumor. At the same time, there are some variables colored with blue that reduce the probability of getting the benign class. Their impact is small, so they are not displayed. The largest effect there is the worst concavity size of 0.2671, while the average for all patients (i.e. including those who have cancer tumor) is 0.2722. It means that the patient has the worst concavity size at the level close to the risk. So, for the benign cases only this factor equals 0.1662, which is less than the average for all patients (0.2722).

Figure 18: SHAP for the test benign case. The prediction is correct.

Let's do the same procedure for the second test case 19. Its prediction is "0", malignant, and this is also the true value. The most impactful factors are: the worst concave points size of 0.1789 (the average is 0.1146), the worst are size of 1866 (the average is 880.5831), and the worst perimeter size of 165.9 (the is 107.26). There are no perceptible effects that reduce the chance of getting the malignant class.

Figure 19: SHAP for the test malignant case. The prediction is correct.

Figure 20 summarizes SHAP results for all cases. It contains an ordered list of 10 most important features, the last one is aggregated as a sum of 21 other factors. Thus, the most crucial variables are worst concave points, mean concave points, and worst area. As can be noticed, the higher feature values indicate that there is less chance of getting benign forecast. Simply, the larger tumor is related to the higher risk of the malignancy.

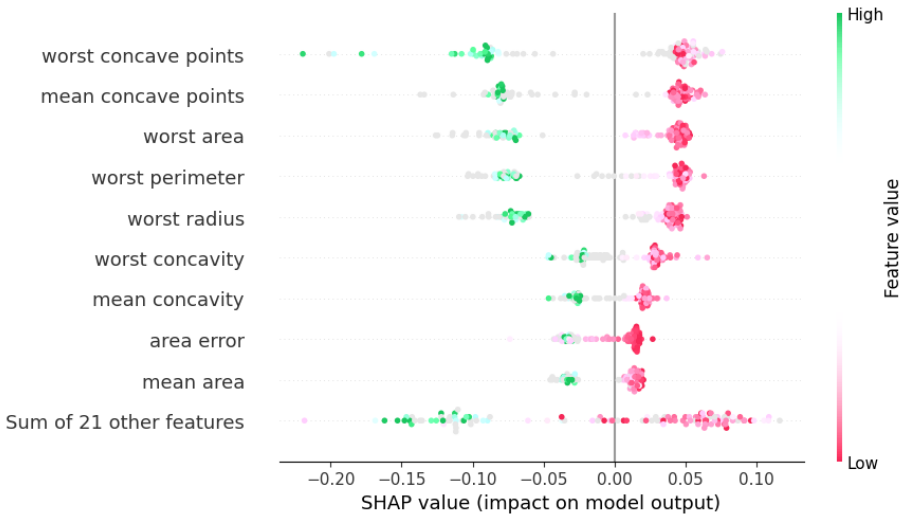

Figure 20: SHAP for the global explanations of the random forest classifier.

Explanations for regression. California housing data analysis results are analysed in this subsection. Let's draw a random instance from the test set and provide SHAP explanations for it21. The real target value for this particular Californian house is 2.186, which is slightly higher than the average price of houses (2.066 abstract units).

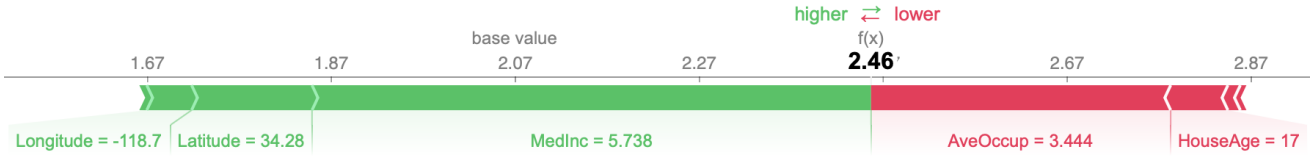

Figure 21: SHAP for the test instance. The real house price equals 2.186.

As for the case of the random forest classifier, it is possible to provide global explanations23. One can notice that high level of median income in block group (MedInc) increases a predicted house price. The tendency flips when we look at the average number of household members (AveOccup). Houses populated with large families are cheaper. Next, both the latitude and longitude are relevant for the predictions. The results show that real estate on the southwest part of California (closer to the coastline) is priced higher. A specific dependence plot22 was created to highlight these relations.

The median house age in the block group (HouseAge) is also important for the model. So, districts occupied mainly by newer houses are cheaper in terms of housing prices. Other factors have less influence on the predictions.

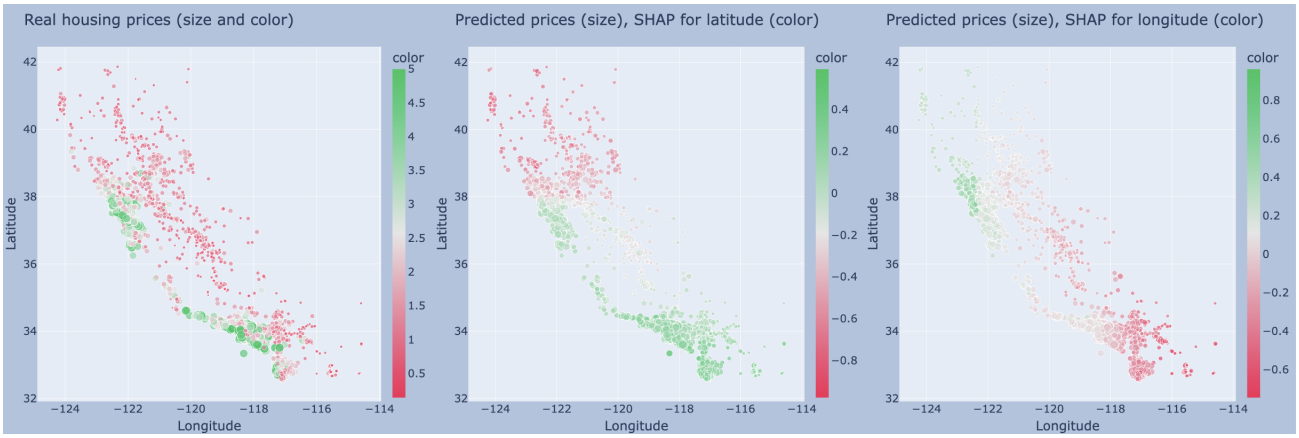

Figure 22: Comparison of real and predicted Californian housing prices, where SHAP values for both the latitude and longitude are indicated.

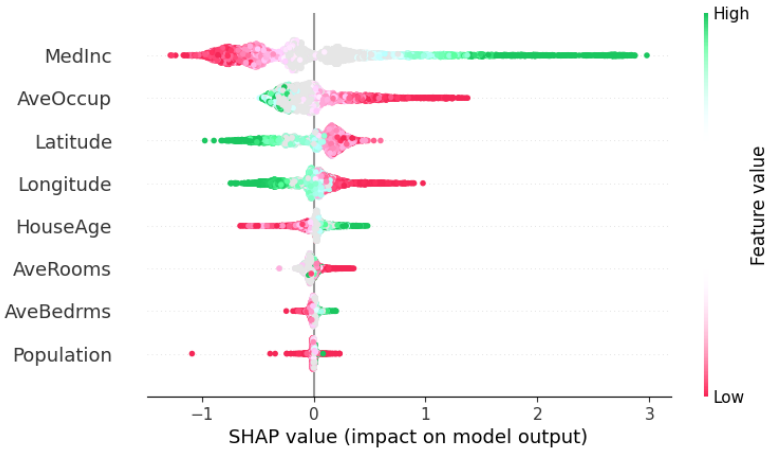

Figure 23: SHAP for the global explanations of the random forest regression.

5.2.2 Implementation of SHAP for imagery data

The main principles of SHAP work well for both tabular and imagery data. Nevertheless, application of SHAP on imagery data has one important limitation: one cannot use SHAP globally for the whole set of predictions to be explained. All the necessary packages and functions are presented in Appendix 8, as for LIME.

Let's start with the use of the same test image, the speed limit of 30 km/h. In this thesis, it was decided to provide segmentation (superpixels) for calculating Shapley values, as a first option. After several trials, the choice was made to use 100 superpixels to get more detailed info on picture's specifics.

Segmentation and calculation of Shapley values delivers five results for every class used in the CNN 24. As can be seen, the result for the most likely class demonstrates that the combination of "3" and, partially, "0" increases overall probability of the class "30km". On the contrary, the lowest predicted probability value is for "20km", where almost every superpixel contributes nothing, they are in grey, and "70km", where several superpixels are in light red (negative impact on the overall likelihood).

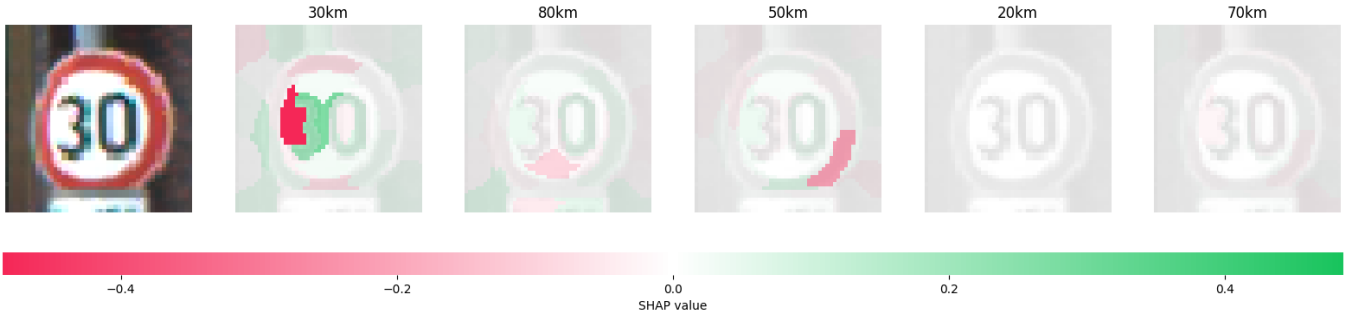

Figure 24: SHAP for the segmented image (SLIC). Classes are arranged in descending order of likelihood, starting with the most likely on the left, which is "30km".

Some researchers also include SHAP for analyzing every particular pixel. That is problematic for high-resolution images. However, it is possible to try, since these photos are only of 50×50 pixels. So, in picture number 25 five possible classes are shown. Based on available evidence, it appears that commenting these results is hard due to the noise and number of dots. However, one can notice such a tendency. The case "50km", the third image, contains too much red pixels, though there are also greed ones. The last picture, "20km", is in grey. Thus, regardless of combinations and changes, the probability of getting this class is low.

In this thesis, the third option of working with images and SHAP for interpretation of the CNN's predictions for them is also considered. So, the images can be split into larger segments, but without any ML tool such as SLIC. Instead, it is possible to dissect images linearly, simply by enlarging several pixels into one. I used this approach for the third variation of SHAP. There were selected random 5 pictures from the test set and implemented linear segmentation26. As one can see, all of the photos are classified correctly. Thus, the most likely class for the first image is "30km", and big segments close the integer "3". The second picture, "priority roads", classified precisely due to the diamond shape in the central part. The third image, which was taken at night

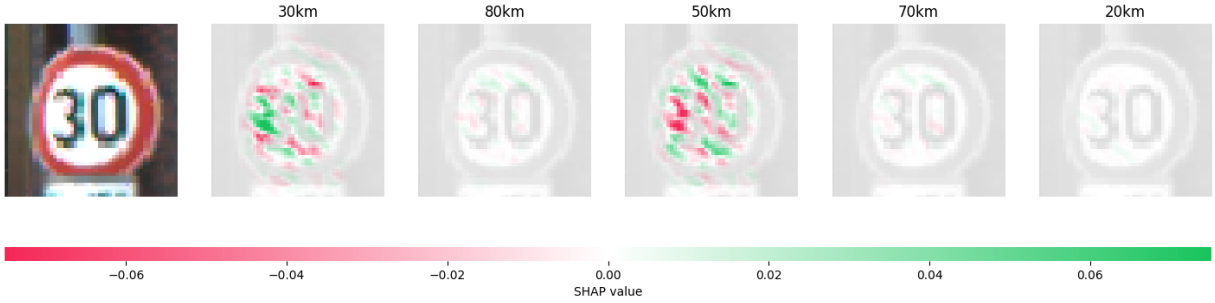

Figure 25: SHAP for the initial image, i.e. without any segmentation.

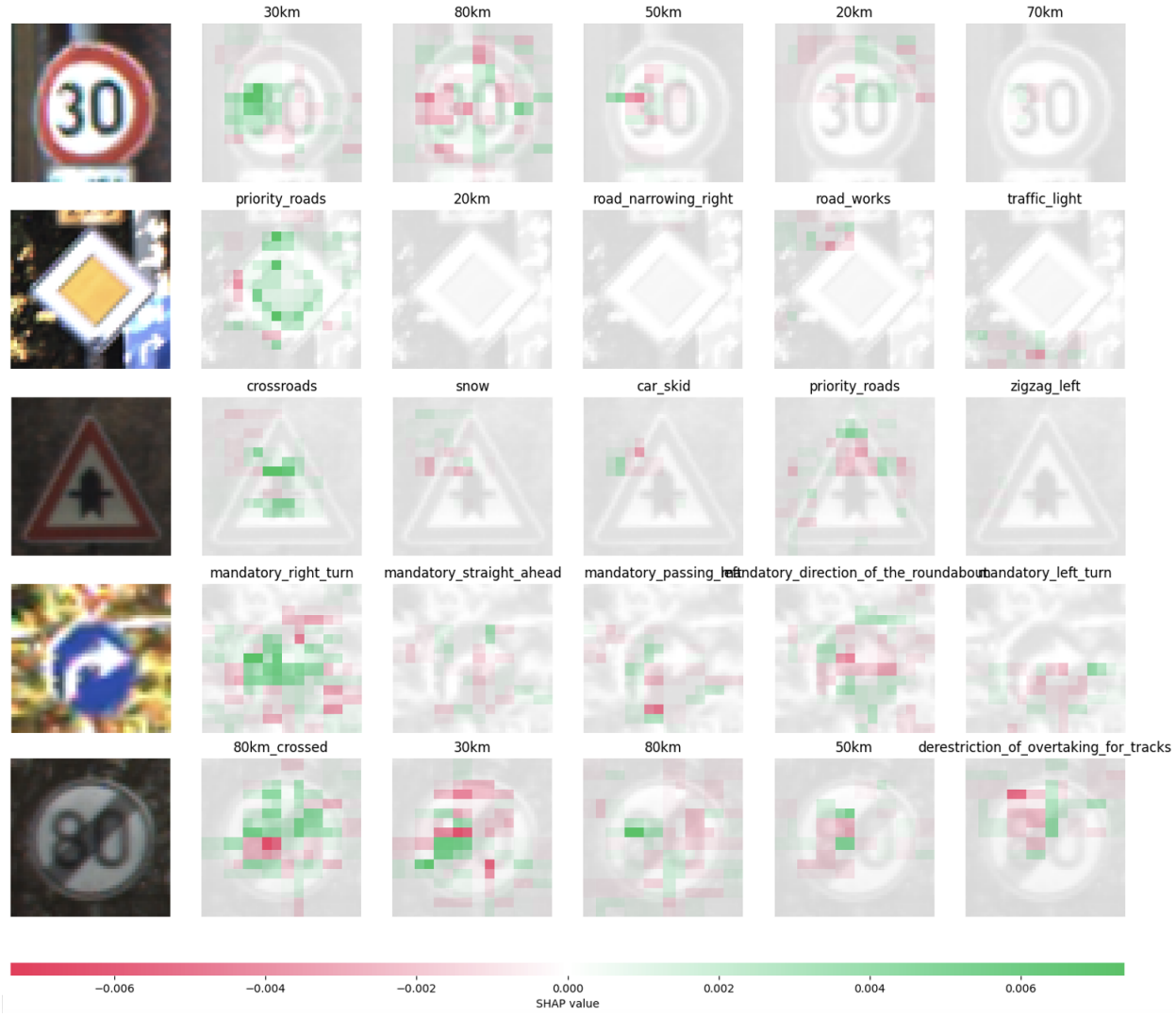

Figure 26: SHAP for the first 5 test images. The case of linear segmentation.

or evening, is labeled as “crossroads”, and the reason is the top part of the central symbol. It can also remind another sign, the “snow”, but its top part is not similar enough to a snowflake. The fourth traffic sign is “mandatory right turn”, which cannot be misclassified due to the white arc on the left side. The last picture is “80km crossed”, which means speed derestriction. This image is dim. The combination of integers and a line in this sign increases the chance of the correct prediction. The general outcome is the fact that the background and image quality and additional specifics did not influence much, and most predictions are correct and explainable. Thus, there is no evidence that under/over-fitting took place in this analysis.

The method of linear segmentation appears to be the most efficient and the fastest one for this data set. As an additional inference derived from this empirical research, predominantly relying on this approach is recommended in case an analyst does not need pixel-by-pixel information.

6 Conclusions

To conclude, interpreting results of black box models should be an important part of a research in almost any area. This component of model quality control, which could be done by means of LIME and SHAP, is absolutely crucial in clinical investigation, computer vision for car automatization, etc. These methods are applicable for financial analysis, real estate business, and in many other topics. So, this thesis showed that the use of machine learning non-linear models in conjunction with posteriori black box interpretations can be beneficial in many areas.

Advantages, disadvantages, and limitations of black box explainers, or Explainable AI, should be summarized. We could start with the simpler mechanism, LIME. Benefits of LIME include the fact that it is designed to interpret the results of any black box model. The initial model structure is unimportant. LIME performs local explanations that focus on particular examples (cross-sectional data rows, images, words in a text, etc.), enabling detailed information on model predictions. The great advantage of using LIME is its simplicity. Using this method, one can expect a quite straightforward application. In principle, all procedures such as image segmentation or creation of a dataset for a linear modeling can be quickly understood and implemented. However, LIME is not flawless and an universal remedy for any case. This algorithm can generate too simplified local explanations. This implies that an analyst employing LIME methodology may unintentionally neglect complexity and non-linearity of an underlying model. So, there is a lack of global perspective. It makes sense, because LIME, by definition, focuses on local explanations and may overlook global patterns and interconnections between variables. Sensitivity to random perturbations is another weakness of LIME. Empirical results of this thesis show that LIME's explanations may be sensitive to small perturbations in the input data. Potentially, it leads to varying or unstable results. So, the solution could be using of more samples to make LIME less unstable.

SHAP is an alternative black box explainer with its pros and cons. Similar to LIME, SHAP is also model agnostic. It and can be used to explain results of numerous models. Additionally, SHAP includes a solid theoretical foundation based on Shapley values from cooperative game theory. Unlike LIME, SHAP is able to provide both global and local explanations, providing a more comprehensive grasp of the model behavior. Nonetheless, SHAP is not a perfect solution for every scenario. One of the most challenging aspects of this algorithm is computational complexity. So, calculating SHAP values can be time-consuming, especially for complex models and high-dimensional data structures.

The common problem of these two methods can be the pre-existing knowledge and beliefs. The interpretation of feature importance in LIME or SHAP may still depend on human interpretation and domain knowledge, introducing potential biases. Furthermore, these algorithms do not offer in-depth comprehension of the model inner structure. Instead, LIME and SHAP provide results at the feature level without uncovering complicated mechanisms and patterns in the hidden layers of the neural networks. Additional note on SHAP is the fact that the linear image segmentation can be considered as the best option among others. So, pixelwise approach is substantially slower and oversophisticated (as a rule, particular pixels has small impact on an overall's prediction). Non-linear segmentation such as SLIC is also unnecessary, because it consumes additional time to slice the image. This technic did not improve the interpretability in the model considered in the empirical part of this thesis.

However, after revising several implementations of both explainers, SHAP seems to be more trustworthy and plausible, especially in case of imagery data. This technique appeared to be more stable that LIME while rerunning the processes, so the same patterns, superpixels, segments were highlighted throughout many program restarts. On the contrary, LIME periodically produced unexpected and random explanations for the CNN classifier. Both LIME and SHAP performed well on the tabular data. Nonetheless, SHAP grants an opportunity to aggregate analysis of the features' influence to provide global explanations. Hence, researchers can examine consolidated information directly. Notwithstanding, it does not imply that LIME may be more efficient and suitable than SHAP for other datasets.

I would also like to add further perspectives of this paper. Firstly, tabular and imagery are not the only data types, there is plenty of others. They can be also used in machine learning models and black box surrogate explainers. As an illustration, SHAP and LIME can be also used for textual information. However, is was described and relatively well-documented before. Personally, the most uncovered area here is a cluster of models for time series data. Up to the moment, there is a lack of theoretical materials on this topic. As a rare exception, there is an article[Gui22], which provides an extensive literature review on black box explainers for time series classification. Authors concluded that the model[Don20] is "promising, uncommon method". There are also some references to black box explainers, which are both global and time-series-specific. I am interested in further exploring of this topic and searching for algorithms for time series analysis, including regression models. Which is also important, such underexplored models are not implemented yet as libraries in any programming language. So it would be important for me to create a real application of the Explainable AI (XAI) for time series.[23]

7 Bibliography

Articles

[Beh10] D. Scherer; A. Müller; S. Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition." In: *International conference on artificial neural networks (pp. 92-101). Berlin, Heidelberg: Springer Berlin Heidelberg.* (2010).

[Bre01] L. Breiman. "Random forests". In: *Machine learning, 45, 5-32.* (2001).

[Bui21] H. Guo; H. Nguyen; D. A. Vu; X. N. Bui. "Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach." In: *Resources Policy, 74, 101474.* (2021).

[Cai18] S. Qiu; X. Xu; B. Cai. "FReLU: flexible rectified linear units for improving convolutional neural networks." In: *In 2018 24th International Conference on Pattern Recognition (ICPR) (pp. 1223-1228). IEEE.* (2018).

[Cre20] B. Williams; C. Halloin; W. Löbel; F. Finklea; E. Lipke; R. Zweigerdt; S. Cremaschi. "Data-driven model development for cardiomyocyte production experimental failure prediction." In: *Computer aided chemical engineering (Vol. 48, pp. 1639-1644). Elsevier.* (2020).

[Don20] S. Mohammadinejad; J. V. Deshmukh; A. G. Puranic; M. Vazquez-Chanlatte; A. Donzé. "Interpretable classification of time-series data using efficient enumerative techniques." In: *In Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control (pp. 1-10).* (2020).

[Gue16] M. T. Ribeiro; S. Singh; C. Guestrin. ""Why should I trust you?" Explaining the predictions of any classifier." In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144)* (2016).

[Gui22] A. Theissler; F. Spinnato; U. Schlegel; R. Guidotti. "Explainable AI for time series classification: a review, taxonomy and research directions". In: *IEEE Access* (2022).

[Ho95] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.* (1995).

[Hus20] N. Jothi; W. Husain. "Predicting generalized anxiety disorder among women using Shapley value". In: *Journal of infection and public health, 14(1), 103-108.* (2020).

[Irv16] Benjamin Irving. "maskSLIC: regional superpixel generation with application to local pathology characterisation in medical images." In: *arXiv:1606.09518* (2016).

[Jac89] Y. LeCun; B. Boser; J.S. Denker; D. Henderson; R.E. Howard; W. Hubbard; L.D. Jackel. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation, 1(4), 541-551.* (1989).

[KB17] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980.* (2017).

[Kon14] E. Štrumbelj; I. Kononenko. "Explaining prediction models and individual predictions with feature contributions." In: *Knowledge and information systems 41.3, 647-665.* (2014).

[Lee17] S. M. Lundberg; S. I. Lee. "A unified approach to interpreting model predictions." In: *Advances in neural information processing systems, 30.* (2017).

[Lek23] A. Salih; Z. Raisi-Estabragh; I. B. Galazzo; P. Radeva; S. E. Petersen; G. Menegaz; K. Lekadir. "Commentary on explainable artificial intelligence methods: SHAP and LIME." In: *arXiv preprint arXiv:2305.02012.* (2023).

[Mez23] A. Stadlhofer; V. Mezhuyev. "Approach to provide interpretability in machine learning models for image classification." In: *Industrial Artificial Intelligence, 1(1), 10.* (2023).

[Neu10] A. N. Spiess; N. Neumeyer. "An evaluation of R2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach." In: *BMC pharmacology, 10(1), 1-11.* (2010).

[Pre02] L. Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade (pp. 55-69). Berlin, Heidelberg: Springer Berlin Heidelberg.* (2002).

[Sal14] N. Srivastava; G. Hinton; A. Krizhevsky; I. Sutskever; R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research, 15(1), 1929-1958.* (2014).

[Sha53] L. S. Shapley. "A value for n-person games". In: *Princeton University Press Princeton* (1953).

[Süs12] Radhakrishna Achanta; Appu Shaji; Kevin Smith; Aurelien Lucchi; Pascal Fua; Sabine Süsstrunk. "SLIC Superpixels Compared to State-of-the-art Superpixel Methods." In: *TPAMI. DOI:10.1109 / TPAMI.2012.120* (2012).

[Yu18] J. Wang; J. Zhang; W. Bao; X. Zhu; B. Cao; P. S. Yu. "Not just privacy: Improving performance of private deep learning in mobile cloud." In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2407-2416).* (2018).

[Zhu21] T. Pearce; A. Brintrup; J. Zhu. "Understanding softmax confidence and uncertainty." In: *arXiv preprint arXiv:2106.04972.* (2021).

[Zis14] K. Simonyan; A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556.* (2014).

Books

[Cou16] I. Goodfellow; Y. Bengio; A. Courville. *Deep learning. MIT press.* 2016. ISBN: 9780262035613.

[Hay09] S. Haykin. *Neural networks and learning machines.* 2009. ISBN: 9780131471399.

[Mol22] C. Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.* 2022. URL: `https://christophm.github.io/interpretable-ml-book`.

[Sla15] R. Baraniuk; T. Antoulas; S. Cox; B. Fite; R. Ha; M. Haag; M. Hutchinson; D. Johnson; R. Radaelli-Sanchez; J. Romberg; P. Schniter; M. Selik; J. P. Slavinsky. *Signals and systems.* Rice University, 2015. ISBN: 9781616100681.

[Sub20] A. Subasi. *Practical machine learning for data analysis using Python. Academic Press.* 2020. ISBN: 978-0-12-821379-7.

[Was96] J. Neter; M. H. Kutner; C. J. Nachtsheim; W. Wasserman. *Applied linear statistical models.* 1996. ISBN: ISBN 0-07-238688-6.

Online Sources

[21] *Proposal for a Regulation of the European Parliament and of the Council. Laying down harmonized rules on Artificial Intelligence (Artificial Intelligence Act) and amending certain Union Legislative Acts.* Accessed on January 25, 2024. 2021. URL: https://artificialintelligenceact.eu/the-act/.

[23] *EU AI Act: first regulation on artificial intelligence.* Accessed on January 25, 2024. 2023. URL: https://www.europarl.europa.eu/news/en/headlines/society/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence.

[Akr] Karin Akre. *F-score, definition.* Accessed on January 25, 2024. URL: https://www.britannica.com/science/F-score.

[Bri] Britannica. *Accuracy, definition.* Accessed on January 25, 2024. URL: https://www.britannica.com/science/accuracy.

[Cos] Cosmas. *Traffic Sign Classifier, an example from GitHub.* Accessed on January 25, 2024. URL: https://github.com/ItsCosmas/Traffic-Sign-Classification.

[Gup23] Shikha Gupta. *Traffic Signs Recognition using CNN and Keras in Python.* Accessed on January 25, 2024. 2023. URL: https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/.

[Ker] Keras. *Keras FAQ.* Accessed on January 25, 2024. URL: https://keras.io/getting_started/faq/.

[LIM] LIME. *LIME package for Python.* Accessed on January 25, 2024. URL: https://lime-ml.readthedocs.io/en/latest/lime.html.

[Mat10] Arthur Mattuck. *Lecture 21: Convolution Formula.* Accessed on January 25, 2024. 2010. URL: https://ocw.mit.edu/courses/18-03-differential-equations-spring-2010/resources/lecture-21-convolution-formula/.

[Mov] Lalith Movva. *99% accuracy on German Traffic Sign Recognition.* Accessed on January 25, 2024. URL: https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition.

[Myk] Mykola. *GTSRB - German Traffic Sign Recognition Benchmark.* Accessed on January 25, 2024. URL: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/data.

[Pér19] Daniel Hugo Cámpora Pérez. *A practical approach to Convolutional Neural Networks.* Accessed on January 25, 2024. 2019. URL: https://indico.cern.ch/event/766995/contributions/3295783/attachments/1802769/2940979/Daniel_Campora_-_A_practical_approach_to_Convolutional_Neural_Networks_slides.pdf.

[Rat21] Phani Ratan. *What is the Convolutional Neural Network Architecture?* Accessed on January 25, 2024. 2021. URL: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/.

[Sac21] Ahmad Sachal. *Traffic Signs Recognition: CNN Model with Tensorflow Serving.* Accessed on January 25, 2024. 2021. URL: https://medium.com/red-buffer/traffic-signs-recognition-cnn-model-with-tensorflow-serving-d2ed249c943e.

[Scia] Scikit-learn. *Mean absolute percentage error, definition.* Accessed on January 25, 2024. URL: https://scikit-learn.org/stable/modules/model_evaluation.html.

[Scib] Scikit-learn. *The copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset.* Accessed on January 25, 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html.

8 Appendix

1. Variables of the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset

1. Mean radius
2. Mean texture
3. Mean perimeter
4. Mean area
5. Mean smoothness
6. Mean compactness
7. Mean concavity
8. Mean concave points
9. Mean symmetry
10. Mean fractal dimension
11. Radius error
12. Texture error
13. Perimeter error
14. Area error
15. Smoothness error
16. Compactness error
17. Concavity error
18. Concave points error
19. Symmetry error
20. Fractal dimension error
21. Worst radius
22. Worst texture
23. Worst perimeter
24. Worst area
25. Worst smoothness
26. Worst compactness
27. Worst concavity
28. Worst concave points
29. Worst symmetry
30. Worst fractal dimension

2. Labels of car signs

1. 20 km
2. 30 km
3. 50 km
4. 60 km
5. 70 km
6. 80 km
7. 80 km crossed
8. 100 km
9. 120 km
10. No overtaking
11. No overtaking for tracks
12. Crossroads
13. Priority roads
14. Give way
15. Stop
16. Road closed
17. No tractor
18. No entry
19. Attention
20. Sharp curve left
21. Sharp curve right
22. Zigzag left
23. Rough road
24. Car skid
25. Road narrowing right
26. Road works
27. Traffic light
28. Crosswalk
29. Children crossing
30. Bike road
31. Snow
32. Wild animal
33. Derestriction
34. Mandatory right turn
35. Mandatory left turn
36. Mandatory straight ahead
37. Mandatory straight ahead or turning right
38. Mandatory straight ahead or turning left
39. Mandatory passing right
40. Mandatory passing left
41. Mandatory direction of the roundabout
42. Derestriction of overtaking
43. Derestriction of overtaking for tracks

3. Proof of Theorem 1.2

This proof was established in [Bre01]. It suffices to show that there is a set of probability zero C on the sequence space $\Theta_1, \Theta_2, ...$ such that outside of C, for all x,

$$\frac{1}{N}\sum_{n=1}^{N} I(h(\Theta_n, x) = j) \to P_\Theta(h(\Theta, x) = j) \tag{30}$$

For a fixed training set and fixed Θ, the set of all x such that $h(\Theta, x) = j$ is a union of hyper-rectangles. For all $h(\Theta, x) = j$ there is only a finite number K of such unions of hyper-rectangles, denoted by $S_1, ...S_K$. Define $\phi(\Theta) = k$ if $x : h(\Theta, x) = j = S_k$. Let N_k be the number of times that $\phi(\Theta_n) = k$ in the first N trials. Then

$$\frac{1}{N}\sum_{n=1}^{N} I(h(\Theta_n, x) = j) = \frac{1}{N}\sum_{k} N_k I(x \in S_k) \tag{31}$$

By the Law of Large Numbers,

$$N_k = \frac{1}{N}\sum_{n=1}^{N} I(\varphi(\Theta_n) = k) \tag{32}$$

converges a.s. to $P_\Theta(\varphi(\Theta) = k)$. Taking unions of all the sets on which convergence does not occur for some value of k gives a set C of zero probability such that outside of C,

$$\frac{1}{N}\sum_{n=1}^{N} I(h(\Theta_n, x) = j) \to \sum_{k} P_\Theta(\varphi(\Theta) = k) I(x \in S_k) \tag{33}$$

The right hand side is $P_\Theta(h(\Theta, x) = j)$.

4. Proof of Theorem 11.2

This proof was established in [Bre01].

$$PE^*(forest) = E_{X,Y}\left[E_\Theta(Y - h(X, \Theta))\right]^2 \tag{34}$$

$$= E_\Theta E_{\Theta'} E_{X,Y}(Y - h(X, \Theta))(Y - h(X, \Theta^{'})) \tag{35}$$

The term on the right in 34 is a covariance and can be written as:

$$E_\Theta E_\Theta^{'}(\rho(\Theta, \Theta^{'}) sd(\Theta) sd(\Theta^{'}) \tag{36}$$

where $sd(\Theta) = \sqrt{E_{X,Y}(Y - h(X, \Theta))^2}$. Define the weighted correlation as:

$$\bar{\rho} = E_\Theta E_\Theta^{'}(\rho(\Theta, \Theta^{'}) sd(\Theta) sd(\Theta^{'}) / (E_\Theta sd(\Theta))^2 \tag{37}$$

Then

$$PE^*(forest) = \bar{\rho}(E_\Theta sd(\Theta))^2 \leq \bar{\rho} PE^*(tree) \tag{38}$$

5. Python's libraries for the random forest models

```
# Importing required libraries for the analysis of tabular data
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import time
from sklearn import metrics
# dataset
from sklearn.datasets import fetch_california_housing
from sklearn.datasets import load_breast_cancer
# Forests
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
# Trees
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegression
# Plotting
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Black box explainers
import shap
import lime
from lime import lime_tabular
```

6. Python's libraries for the convolutional neural networks

```
# Importing required libraries for the analysis of imagery data
# Operating system dependent functionality
import os
# Data processing
import numpy as np
import pandas as pd
# Plotting
import plotly.express as px
import plotly.subplots as sp
# Image processing
import cv2 as cv
from PIL import Image
# Keras
from keras.models import Sequential
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from keras.models import load_model
from keras.utils import to_categorical
# Quality checking
from sklearn.metrics import accuracy_score
```

7. Data processing: initialization of the framework

```
# based on
    https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
# Reading the input images and putting them into a numpy array
data = []
labels = []
height = 50
width = 50
channels = 3
classes = 43
n_inputs = height * width * channels

# I also included this part to simplify interpreting the results
my_information = {
                0: '20km', 1: '30km',
                2: '50km', 3: '60km',
                4: '70km', 5: '80km',
                6: '80km_crossed', 7: '100km',
                8: '120km', 9: 'no_overtaking',
                10: 'no_overtaking_for_tracks', 11: 'crossroads',
                12: 'priority_roads', 13: 'give_way',
                14: 'stop', 15: 'road_closed',
                16: 'no_tractor', 17: 'no_entry',
                18: 'attention', 19: 'sharp_curve_left',
                20: 'sharp_curve_right', 21: 'zigzag_left',
                22: 'rough_road', 23: 'car_skid',
                24: 'road_narrowing_right', 25: 'road_works',
                26: 'traffic_light', 27: 'crosswalk',
                28: 'children_crossing', 29: 'bike_road',
                30: 'snow', 31: 'wild_animal',
                32: 'derestriction', 33: 'mandatory_right_turn',
                34: 'mandatory_left_turn', 35: 'mandatory_straight_ahead',
                36: 'mandatory_straight_ahead_or_turning_right', 37:
                    'mandatory_straight_ahead_or_turning_left',
                38: 'mandatory_passing_right', 39: 'mandatory_passing_left',
                40: 'mandatory_direction_of_the_roundabout', 41:
                    'derestriction_of_overtaking',
                42: 'derestriction_of_overtaking_for_tracks'}
```

8. Black box explainers for results of CNNs in Python

```
from keras.models import load_model
from keras.preprocessing.image import load_img, img_to_array
import cv2 as cv
from PIL import Image
import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
```

```
from skimage.segmentation import slic, mark_boundaries
from matplotlib.colors import LinearSegmentedColormap
import warnings
import time
# Explainers:
import shap
from keras.applications.resnet50 import preprocess_input
# LIME: libraries for the code from https://lime-ml.readthedocs.io/en/latest/lime.html
from skimage import segmentation
from keras.preprocessing import image as image_keras
import copy
from functools import partial
import scipy as sp
from sklearn.linear_model import Ridge, lars_path
from sklearn.utils import check_random_state
import sklearn
from skimage.color import gray2rgb
from tqdm.auto import tqdm
import types
from lime.utils.generic_utils import has_arg
```

9. Decision tree. Classification for breast cancer data

This model was provided for comparison with the results of the random forest classifier. This is an additional check.

```
clf = DecisionTreeClassifier(max_depth=100)
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
decision_tree_preds_classifier = clf.predict(X_test)
print('The accuracy of the Decision Tree classifier is
    :\t',metrics.accuracy_score(decision_tree_preds_classifier,y_test))
print('The precision of the Decision Tree classifier is
    :\t',metrics.precision_score(decision_tree_preds_classifier,y_test))
print('The recall of the Decision Tree classifier is
    :\t',metrics.recall_score(decision_tree_preds_classifier,y_test))
print('The F1 score of the Decision Tree classifier is
    :\t',metrics.f1_score(decision_tree_preds_classifier,y_test))

plt.figure(figsize=(15,15)) # set plot size (denoted in inches)
plot_tree(clf, filled=True, feature_names=list(X_test.columns),fontsize=6)
plt.show()
```

```
The accuracy of the Decision Tree classifier is :  0.9298245614035088
The precision of the Decision Tree classifier is :  0.9295774647887324
The recall of the Decision Tree classifier is :  0.9565217391304348
The F1 score of the Decision Tree classifier is :  0.9428571428571428
```

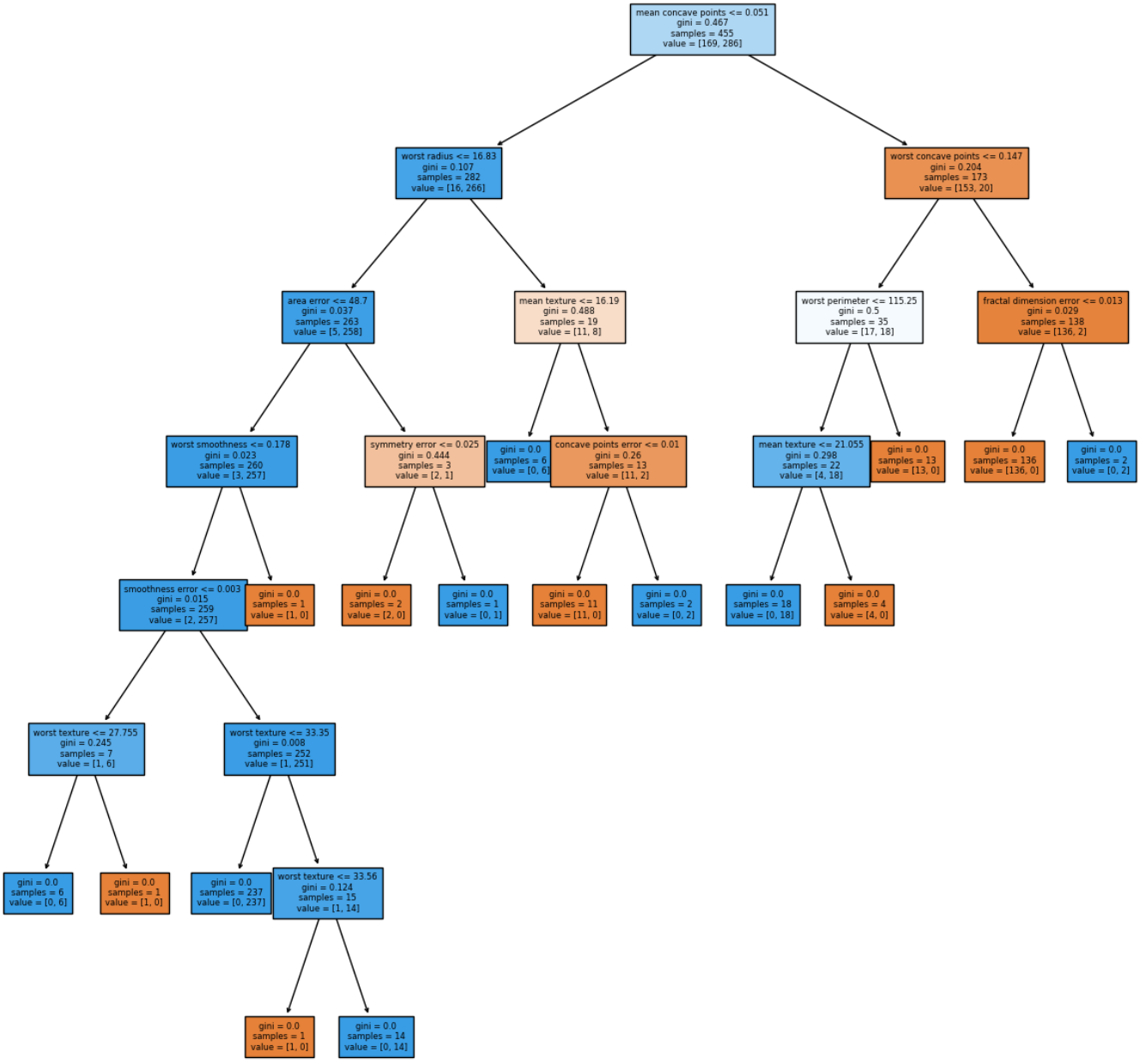

Figure 27: Results of the decision tree classifier.

10. Decision tree. Regression for California housing data

This model was provided for comparison with the results of the random forest regression. This is an additional check.

```
rgr = DecisionTreeRegressor(max_depth=10, max_features=None)
# Train Decision Tree Regressor
rgr = rgr.fit(X_train,y_train)
#Predict the response for test dataset
decision_tree_preds_regression = rgr.predict(X_test)
# Model's quality
print('The MSE of the Decision Tree resgression is
    :\t',metrics.mean_squared_error(decision_tree_preds_regression,y_test))
print('The RMSE of the Decision Tree resgression is
    :\t',(metrics.mean_squared_error(decision_tree_preds_regression,y_test))**0.5)
print('The MAPE of the Decision Tree resgression is
    :\t',metrics.mean_absolute_percentage_error(decision_tree_preds_regression,y_test))
print('The MAE of the Decision Tree resgression is
    :\t',metrics.mean_absolute_error(decision_tree_preds_regression,y_test))
plt.figure(figsize=(15,15)) # set plot size (denoted in inches)
plot_tree(rgr, filled=True, feature_names=list(X_test.columns),fontsize=6)
plt.show()
```

```
The MSE of the Decision Tree resgression is :  0.41834769454342435
The RMSE of the Decision Tree resgression is :  0.6467980322661969
The MAPE of the Decision Tree resgression is :  0.2254281166087855
The MAE of the Decision Tree resgression is :  0.4330337333874766
```

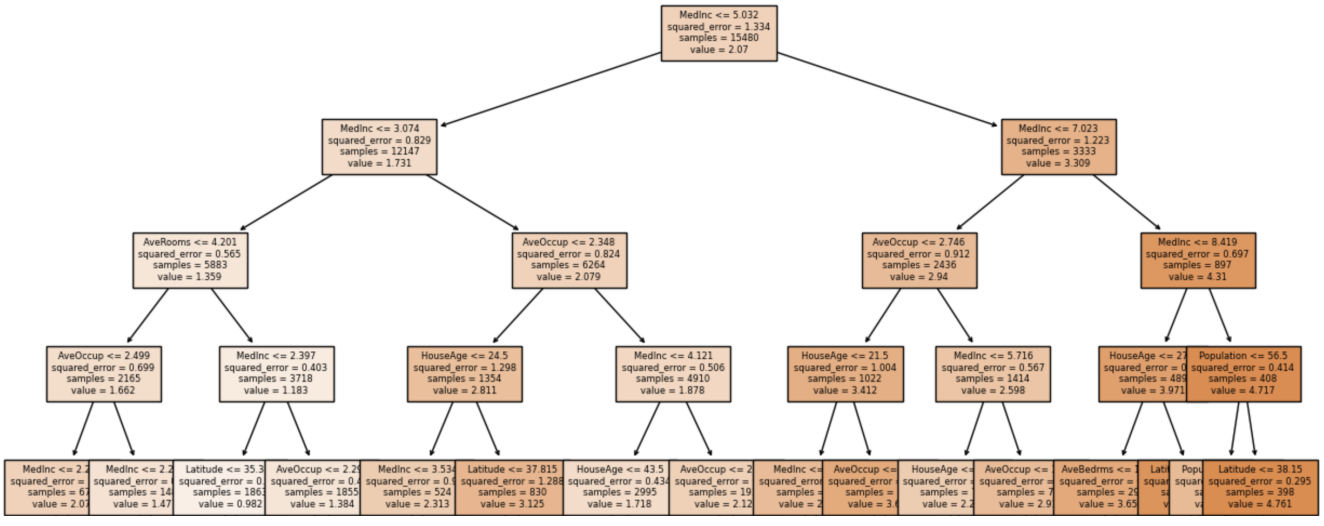

Figure 28: Results of the decision tree regression. Only a fragment is shown due to the model depth which equals to 10.