RuleXAI

Release v1.0.0

Macha Dawid

Jul 08, 2022

CONTENTS:

Welcome to RuleXAI's documentation!

RuleXAI is a rule-based aproach to explain the output of any machine learning model. It is suitable for classification, regression and survival tasks. Theoretical basis of the rule analysis methods implemented in the RuleXAI package can be found in Theoretical basis section.

CHAPTER

ONE

INSTALLATION

RuleXAI can be installed from PyPI:

```
pip install rulexai
```

1.1 Theoretical basis

Click here to view document describing theoretical basis of the rule analysis methods implemented in the RuleXAI package

1.2 Code documentation

class rulexai.explainer.**RuleExplainer**(*model*, *X: DataFrame*, *y: Union[DataFrame, Series]*, *type: str = 'classification'*)

Parameters

- **model** (*Model = Union[RuleClassifier, RuleRegressor, SurvivalRules, CN2UnorderedClassifier, CN2SDUnorderedClassifier, DecisionTreeClassifier, DecisionTreeRegressor, SurvivalTree, List[str]]*) –

  Model to be analyzed. RuleXai supports the following Rule models:

  – RuleKit(https://adaa-polsl.github.io/RuleKit-python/): RuleClassifier, RuleRegressor, SurvivalRules

  – Orange (https://orangedatamining.com/): CN2UnorderedClassifier, CN2SDUnorderedClassifier

  It can also extract rules from decision trees:

  – scikit-learn (https://scikit-learn.org/stable/): DecisionTreeClassifier, DecisionTreeRegressor

  – scikit-survival (https://scikit-survival.readthedocs.io/en/stable/): SurvivalTree

  Or you can provide a list of rules as:

  – **classification:**
    IF attribute1 = (-inf, value) AND … AND attribute2 = <value1, value2) THEN label_atrribute = {class_name}

  - **regression:**
    IF attribute1 = (-inf, value) AND … AND attribute2 = <value1, value2) THEN target_attribute = {value}
  - **survival:**
    IF attribute1 = (-inf, value) AND … AND attribute2 = <value1, value2) THEN survival_status_attribute = {survival_status}

- **X** (*pd.DataFrame*) – The training dataset used during provided model training
- **y** (*Union[pd.DataFrame, pd.Series]*) – The target values (class labels, real number, survival status) used during provided model training
- **type** (*str = None*) –

  The type of problem that the provided model solves. You can choose between:

  - "classification"
  - "regression"
  - "survival"

  default: "classification"

condition_importances_

Computed conditions importances

Type
pd.DataFrame

feature_importances_

Feature importances computed base on conditions importances

Type
pd.DataFrame

explain(*measure: str = 'C2', basic_conditions: bool = False*)

Compute conditions importances. The importances of a conditions are computed base on:

Marek Sikora: Redefinition of Decision Rules Based on the Importance of Elementary Conditions Evaluation. Fundam. Informaticae 123(2): 171-197 (2013)

https://dblp.org/rec/journals/fuin/Sikora13.html

Parameters

- **measure** (*str*) – Specifies the measure that is used to evaluate the quality of the rules. Possible measures for classification and regression problem are: C2, Lift, Correlation. Default: C2. It is not possible to select a measure for the survival problem, the LogRank test is used by default
- **basic_conditions** (*bool*) – Specifies whether to evaluate the conditions contained in the input rules, or to break the conditions in the rules into base conditions so that individual conditions do not overlap

Returns
self – Fitted explainer with calculated conditions

Return type
Explainer

fit_transform(*X: DataFrame*, *selector=None*, *y=None*, *POS=None*) → DataFrame

Creates a dataset based on given dataset in which the examples, instead of being described by the original attributes, will be described with the specified conditions - it will be a set with binary attributes determining whether a given example meets a given condition. It can be considered as kind of dummification. Thanks to this function you can discretize data and get rid of missing values. It can be used as prestep for others algorithms.

Parameters

- **X** (*pd.DataFrame*) – The input samples from which you want to create binary dataset. Should have the same columns and columns order as X specified when creating Explainer
- **selector** (*string/float*) – Specifies on what basis to select the conditions from the rules that will be included as attributes in the transformed set. If None all conditions will be included in the transformed set. If number 0-1 percent of the most important conditions will be selected based on condition importance ranking. If "reduct" the reduct of the conditions set will be selected. Preferably, the option with the percentage of most important conditions will be selected.
- **y** (*Union[pd.DataFrame, pd.Series]*) – Only if selector = "reduct".The target values for input sample, used in the determination of the reduct
- **POS** (*float*) – Only if selector = "reduct".Target reduct POS

Returns
X_transformed – Transformed dataset

Return type
pd.DataFrame

get_rules()

Return rules from model

Returns
rules – Rules from model

Return type
List[str]

get_rules_covering_example(*x: DataFrame*, *y: Union[DataFrame, Series]*) → List[str]

Return rules that covers the given example

Parameters

- **x** (*pd.DataFrame*) – The input sample.
- **y** (*Union[pd.DataFrame, pd.Series]*) – The target values for input sample.

Returns
rules – Rules that covers the given example

Return type
List[str]

get_rules_with_basic_conditions()

Return rules from model with conditions broken down into base conditions so that individual conditions do not overlap

Returns
rules – Rules from the model containing the base conditions

Return type
List[str]

`local_explainability`(*x: DataFrame*, *y: Union[DataFrame, Series]*, *plot: bool = False*)

Displays information about the local explanation of the example: the rules that cover the given example and the importance of the conditions contained in these rules

Parameters

- **x** (*pd.DataFrame*) – The input sample.
- **y** (*Union[pd.DataFrame, pd.Series]*) – The target values for input sample.
- **plot** (*bool*) – If True the importance of the conditions will also be shown in the chart. Default: False

`plot_importances`(*importances: DataFrame*)

Plot importances :param importances: Feature/Condition importances to plot. :type importances: pd.DataFrame

`transform`(*X: DataFrame*) → DataFrame

Creates a dataset based on given dataset in which the examples, instead of being described by the original attributes, will be described with the specified conditions - it will be a set with binary attributes determining whether a given example meets a given condition. It can be considered as kind of dummification. Thanks to this function you can discretize data and get rid of missing values. It can be used as prestep for others algorithms.

Parameters
X (*pd.DataFrame*) – The input samples from which you want to create binary dataset. Should have the same columns and columns order as X given in fit_transform

Returns
X_transformed – Transformed dataset

Return type
pd.DataFrame

class rulexai.explainer.**Explainer**(*X: DataFrame*, *model_predictions: Union[DataFrame, Series]*, *type: str = 'classification'*)

Parameters

- **X** (*pd.DataFrame*) – The training dataset used during provided model training
- **model_predictions** (*Union[pd.DataFrame, pd.Series]*) – The training dataset used during provided model training
- **type** (*str*) –

  The type of problem that the provided model solves. You can choose between:

  – ”classification”

  – ”regression”

  default: “classification”

`condition_importances_`

Computed conditions importances on given dataset

Type
pd.DataFrame

`feature_importances_`

Feature importances computed base on conditions importances

Type
pd.DataFrame

`explain`(*measure: str = 'C2'*, *basic_conditions: bool = False*, *X_org=None*)

Compute conditions importances. The importances of a conditions are computed base on:

Marek Sikora: Redefinition of Decision Rules Based on the Importance of Elementary Conditions Evaluation. Fundam. Informaticae 123(2): 171-197 (2013)

https://dblp.org/rec/journals/fuin/Sikora13.html

Parameters

- `measure` (*str*) – Specifies the measure that is used to evaluate the quality of the rules. Possible measures for classification and regression problem are: C2, Lift, Correlation. Default: C2. It is not possible to select a measure for the survival problem, the LogRank test is used by default
- `basic_conditions` (*bool*) – Specifies whether to evaluate the conditions contained in the input rules, or to break the conditions in the rules into base conditions so that individual conditions do not overlap
- `X_org` – The dataset on which the rule-based model should be built. It can be the set on which the black-box model was learned or this set before preprocessing (imputation of missing values, dummification, scaling), because such a set can be handled by the rule model

Returns
self – Fitted explainer with calculated conditions

Return type
Explainer

`fit_transform`(*X: DataFrame*, *selector=None*, *y=None*, *POS=None*) → DataFrame

Creates a dataset based on given dataset in which the examples, instead of being described by the original attributes, will be described with the specified conditions - it will be a set with binary attributes determining whether a given example meets a given condition. It can be considered as kind of dummification. Thanks to this function you can discretize data and get rid of missing values. It can be used as prestep for others algorithms.

Parameters

- `X` (*pd.DataFrame*) – The input samples from which you want to create binary dataset. Should have the same columns and columns order as X specified when creating Explainer
- `selector` (*string/float*) – Specifies on what basis to select the conditions from the rules that will be included as attributes in the transformed set. If None all conditions will be included in the transformed set. If number 0-1 percent of the most important conditions will be selected based on condition importance ranking. If "reduct" the reduct of the conditions set will be selected. Preferably, the option with the percentage of most important conditions will be selected.
- `y` (*Union[pd.DataFrame, pd.Series]*) – Only if selector = "reduct".The target values for input sample, used in the determination of the reduct
- `POS` (*float*) – Only if selector = "reduct".Target reduct POS

Returns
X_transformed – Transformed dataset

Return type
pd.DataFrame

`get_rules()`

Return rules from model

Returns
rules – Rules from model

Return type
List[str]

`get_rules_covering_example`(*x: DataFrame*, *y: Union[DataFrame, Series]*) → List[str]

Return rules that covers the given example

Parameters

- **x** (*pd.DataFrame*) – The input sample.
- **y** (*Union[pd.DataFrame, pd.Series]*) – The target values for input sample.

Returns
rules – Rules that covers the given example

Return type
List[str]

`get_rules_with_basic_conditions()`

Return rules from model with conditions broken down into base conditions so that individual conditions do not overlap

Returns
rules – Rules from the model containing the base conditions

Return type
List[str]

`local_explainability`(*x: DataFrame*, *y: Union[DataFrame, Series]*, *plot: bool = False*)

Displays information about the local explanation of the example: the rules that cover the given example and the importance of the conditions contained in these rules

Parameters

- **x** (*pd.DataFrame*) – The input sample.
- **y** (*Union[pd.DataFrame, pd.Series]*) – The target values for input sample.
- **plot** (*bool*) – If True the importance of the conditions will also be shown in the chart. Default: False

`plot_importances`(*importances: DataFrame*)

Plot importances :param importances: Feature/Condition importances to plot. :type importances: pd.DataFrame

`transform`(*X: DataFrame*) → DataFrame

Creates a dataset based on given dataset in which the examples, instead of being described by the original attributes, will be described with the specified conditions - it will be a set with binary attributes determining whether a given example meets a given condition. It can be considered as kind of dummification. Thanks to this function you can discretize data and get rid of missing values. It can be used as prestep for others algorithms.

Parameters
: **X** (*pd.DataFrame*) – The input samples from which you want to create binary dataset. Should have the same columns and columns order as X given in fit_transform

Returns
: **X_transformed** – Transformed dataset

Return type
: pd.DataFrame

1.3 Tutorials

1.3.1 RuleXAI

In this notebook, the data from https://www.kaggle.com/c/titanic is analysed to show the advantages and possibilities of using the RuleXAI library for in-depth analysis of the dataset. It is a popular set, often used in various types of examples, therefore it was decided to use it in this analysis.

Overview

```
I. Initial data analysis and preprocesing
II. Use of a decision tree from sklearn
III. Analysis of the decision tree model from the previous point with RuleXAI
IV. Using the RuleKit library - a versatile tool for rule learning - to generate rule
V. Analysis with RuleXAI of rules derived with RuleKit
VI. Summary
```

I. Initial data analysis and preprocesing

1. Data load

The data used in this analysis comes from the kaggle competition (https://www.kaggle.com/c/titanic). Two datasets were published as part of this competition:
- training set (train.csv)
- test set (test.csv)

According to the competition rules: "The training set should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use feature engineering to create new features. The test set should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic."

As the purpose of this analysis is to present the RuleXAI library not to take part in the competition, it was decided to use only the data contained in the training set in the further analysis. Therefore, the data from the train.csv file can be split into training and test data, so that it will be possible to evaluate the results obtained without participating in the competition.

[1]:
```
import pandas as pd
```

[4]:
```
dataset_path = "./data/titanic_kaggle.csv"
data = pd.read_csv(dataset_path)
data.head(5)
```

[4]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

2. Dataset overwiev

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

[4]:
```
data[['PassengerId', 'Survived', 'Pclass']
     ] = data[['PassengerId', 'Survived', 'Pclass']].astype(str)
```

```
[5]: numeric_data = data[['Age', 'SibSp', 'Parch', 'Fare']]
     caterogical_data = data[['PassengerId', 'Survived', 'Pclass',
                              'Sex', 'Ticket', 'Cabin', 'Embarked']]
```

```
[6]: numeric_data.describe()
```

```
[6]:
              Age       SibSp       Parch        Fare
count  714.000000  891.000000  891.000000  891.000000
mean    29.699118    0.523008    0.381594   32.204208
std     14.526497    1.102743    0.806057   49.693429
min      0.420000    0.000000    0.000000    0.000000
25%     20.125000    0.000000    0.000000    7.910400
50%     28.000000    0.000000    0.000000   14.454200
75%     38.000000    1.000000    0.000000   31.000000
max     80.000000    8.000000    6.000000  512.329200
```

```
[7]: caterogical_data.describe()
```

```
[7]:
       PassengerId Survived Pclass   Sex    Ticket        Cabin Embarked
count          891      891    891   891       891          204      889
unique         891        2      3     2       681          147        3
top            675        0      3  male  CA. 2343  C23 C25 C27        S
freq             1      549    491   577         7            4      644
```

3. Data preprocessing

In the first stage of data preprocessing it was decided to only remove the columns for PassengerId, Passenger Name, Ticket type and Cabin. Removing the PassengerId and Passenger Name columns is self-explanatory - in no way does PassengerId or Passenger Name have any bearing on whether a person survived. It would only be possible to derive passenger status from passenger name, as there are markings such as 'Mr.', 'Mrs.', 'Miss.', 'Master.'. In case of tickets, the designations for most tickets vary - 681 unique values out of 891 occurrences. One could extract some information from the tickets from their designations (e.g., whether they begin with a number or a letter). However, historical data would need to be consulted to find out what the ticket designations mean. In the case of cabin designations, as many as 697 values are missing - for this reason it was decided to remove the entire column, as it carries too little information.

Of course, the preliminary data analysis and preprocessing stage itself could have been even more extensive - exploring the relationships between features, examining the impact of individual features, plotting graphs to better understand the data. However, the main purpose of this notebook is not to analyse a given set of data in detail, but only to show the possibilities of using the RuleXAI library. For this reason, some simplifications in the analysis have been decided.

```
[8]: data.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1, inplace=True)
     data.reset_index(inplace=True, drop=True)
     data.head(5)
```

```
[8]:
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         0       3    male  22.0      1      0   7.2500        S
1         1       1  female  38.0      1      0  71.2833        C
2         1       3  female  26.0      0      0   7.9250        S
3         1       1  female  35.0      1      0  53.1000        S
4         0       3    male  35.0      0      0   8.0500        S
```

II. Use of a decision tree from sklearn

In the first stage it was decided to use the decision tree for classification, which is available in the sklearn package and which is also supported by the RuleXAI library.

1. Data preparation for decision tree

Since the decision tree algorithm does not support missing values and only operates on numeric data, it was necessary to fill in missing values (for numeric data the median was used, and for categorical data the mode, which is the most frequent value) and dummify. The numerical data could also be rescaled - however, it was decided not to do so to facilitate further analysis, which will be seen later.

[9]:
```
data.Age = data.Age.fillna(data.Age.median())
data.Embarked = data.Embarked.fillna(data.Embarked.mode())

data_dummies = pd.get_dummies(data.drop(["Survived"], axis=1))
data_dummies_scaled = data_dummies.copy()

X = data_dummies_scaled
y = data.Survived
```

2. Data split for training and test datasets

[10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, )
```

3. Building and testing the model

A simple decision tree model with default parameters was used, since the main goal is not to get the best possible results, but only to show the use of the RuleXAI library.

[11]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import balanced_accuracy_score
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=1, max_depth=5)
cv = cross_val_score(dt, X_train, y_train, cv=5, scoring = "balanced_accuracy")
print(cv)
print(cv.mean())
```

```
[0.69121658 0.82526142 0.73472757 0.72491639 0.8157748 ]
0.7583793541088122
```

[12]:
```
dt.fit(X_train, y_train)
```

[12]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

[13]:
```
balanced_accuracy_score(y_test, dt.predict(X_test))
```

[13]:
```
0.7807425259654559
```

4. Determination of the feature importance from the decision tree in the sklearn package

[14]:
```
features_importances = pd.DataFrame(columns=["Feature", "Importance"])
features_importances.Feature = X_train.columns
features_importances.Importance = dt.feature_importances_
features_importances = features_importances.sort_values(
    by=["Importance"], ascending=False)
features_importances.reset_index(inplace=True, drop=True)

features_importances
```

[14]:

	Feature	Importance
0	Sex_female	0.516327
1	Pclass_3	0.158578
2	Age	0.120043
3	Fare	0.119806
4	SibSp	0.041863
5	Embarked_C	0.028040
6	Pclass_1	0.006695
7	Pclass_2	0.006262
8	Parch	0.002385
9	Sex_male	0.000000
10	Embarked_Q	0.000000
11	Embarked_S	0.000000

Looking at the importance of the features obtained directly from the decision tree in sklearn, it can be seen that the feature that most distinguished whether someone survived or not was the gender of the person, and more precisely whether the person was a woman or not. However, there is no information on whether the fact that a person was a woman caused survival or death. Knowing the context of the data, it can be concluded that men were gentlemen and they let women go first, therefore it can be concluded that if the person was a woman, she had a better chance of survival. The second most important feature that distinguished between survivors and non-survivors was whether the person was traveling in 3rd class. Assuming that the women were saved first, the next selection criterion would be in which class someone traveled. If it was 3rd class, it can be concluded that he was saved last, so there is a high probability that he did not survive. Another feature distinguishing whether or not a person survived was the age of the person. It can be assumed that children and the elderly people were saved from people in their prime, who had a better chance of surviving in difficult conditions. Another important feature was the amount of fare - it can be concluded that people who paid more for the ticket, traveled in a better class and in better conditions. This coincides with the fact that the survivors and the non-survivors also distinguished very well whether they were traveling in 3rd class or not.

We can see that the obtained validities of the features coincide with what can be inferred from the historical data and the context of the event. However, without the context, it would be difficult to conclude which feature indicates which class - on the basis of the presented importance of features, one could only obtain information about the separation of classes, and not about which class a given feature indicates.

5. Model generation based on top 50% of features

[15]:
```
features_number = 0.5 * features_importances.shape[0]
```

[16]:
```
X_train_sklearn_features = X_train[features_importances.loc[0:features_number-1, "Feature
↪"]]
X_test_sklearn_features = X_test[features_importances.loc[0:features_number-1, "Feature
↪"]]

X_train_sklearn_features
```

[16]:
```
     Sex_female  Pclass_3   Age      Fare  SibSp  Embarked_C
445           0         0   4.0   81.8583      0           0
650           0         1  28.0    7.8958      0           0
172           1         1   1.0   11.1333      1           0
450           0         0  36.0   27.7500      1           0
314           0         0  43.0   26.2500      1           0
..          ...       ...   ...       ...    ...         ...
106           1         1  21.0    7.6500      0           0
270           0         0  28.0   31.0000      0           0
860           0         1  41.0   14.1083      2           0
435           1         0  14.0  120.0000      1           0
102           0         0  21.0   77.2875      0           0

[623 rows x 6 columns]
```

[20]:
```
dt_sklearn_features = DecisionTreeClassifier(random_state=1, max_depth=5)
cv = cross_val_score(dt_sklearn_features,
                     X_train_sklearn_features, y_train, cv=5, scoring = "balanced_
↪accuracy")
print(cv)
print(cv.mean())
```

```
[0.74440807 0.83791965 0.73018712 0.7703456  0.79598662]
0.7757694124203101
```

[21]:
```
dt_sklearn_features.fit(X_train_sklearn_features, y_train)
```

[21]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

[22]:
```
balanced_accuracy_score(y_test, dt_sklearn_features.predict(X_test_sklearn_features))
```

[22]:
```
0.7839272393412521
```

Looking at the results obtained on the training set (in cross-validation) and the test set, one can see that selecting only the most important features according to the ranking obtained with sklearn improved the results obtained. Indeed, the selected features have the greatest impact in distinguishing whether a person survived or not.

III. Analysis of the decision tree model from the previous point with RuleXAI

1. RuleXAI initialisation

[23]:
```
from rulexai.explainer import RuleExplainer
explainer = RuleExplainer(model=dt, X=X_train, y=y_train, type="classification")
explainer.explain()
```

[23]:
```
<rulexai.explainer.RuleExplainer at 0x29d6719f8b0>
```

2. Presentation of the rules derived from the decision tree

[24]:
```
rules = explainer.get_rules()
for rule in rules:
    print(rule)
```

```
IF Sex_female = (-inf, 0.5> AND Age = (6.5, 77.0> AND Fare = (-inf, 52.277099609375> AND␣
↪Pclass_1 = (-inf, 0.5> THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (2.5, 49.5> AND Fare = (-
↪inf, 149.035400390625> THEN Survived = {1}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (-inf, 15.
↪372900009155273> AND Age = (-inf, 36.5> THEN Survived = {1}
IF Sex_female = (-inf, 0.5> AND Age = (6.5, 77.0> AND Fare = (-inf, 52.277099609375> AND␣
↪Pclass_1 = (0.5, inf) THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (15.372900009155273, 23.
↪350000381469727> AND Age = (-inf, 36.5> THEN Survived = {1}
IF Sex_female = (-inf, 0.5> AND Age = (6.5, inf) AND Fare = (59.08749961853027, inf) AND␣
↪Embarked_C = (-inf, 0.5> THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (23.350000381469727,␣
↪inf) AND Age = (7.0, inf) THEN Survived = {0}
IF Sex_female = (-inf, 0.5> AND Age = (6.5, inf) AND Fare = (59.08749961853027, inf) AND␣
↪Embarked_C = (0.5, inf) THEN Survived = {1}
IF Sex_female = (-inf, 0.5> AND Age = (-inf, 6.5> AND SibSp = (-inf, 3.0> THEN Survived␣
↪= {1}
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (49.5, inf) AND Fare = (-
↪inf, 149.035400390625> THEN Survived = {1}
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (2.5, inf) AND Fare =␣
↪(152.5062484741211, inf) THEN Survived = {1}
IF Sex_female = (-inf, 0.5> AND Age = (22.0, inf) AND Fare = (52.277099609375, 59.
↪08749961853027> THEN Survived = {1}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (-inf, 23.
↪350000381469727> AND Age = (36.5, inf) THEN Survived = {0}
IF Sex_female = (-inf, 0.5> AND Age = (-inf, 6.5> AND SibSp = (3.0, inf) AND Parch = (-
↪inf, 1.5> THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (23.350000381469727,␣
↪inf) AND Age = (-inf, 3.5> THEN Survived = {0}
IF Sex_female = (-inf, 0.5> AND Age = (-inf, 6.5> AND SibSp = (3.0, inf) AND Parch = (1.
↪5, inf) AND Fare = (31.331250190734863, inf) THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (-inf, 2.5> AND Pclass_2␣
↪= (0.5, inf) THEN Survived = {1}
IF Sex_female = (0.5, inf) AND Pclass_3 = (0.5, inf) AND Fare = (23.350000381469727,␣
↪inf) AND Age = (3.5, 7.0> THEN Survived = {1}
```

(continues on next page)

(continued from previous page)

```
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (2.5, inf) AND Fare =␣
↪(149.035400390625, 152.5062484741211> THEN Survived = {0}
IF Sex_female = (-inf, 0.5> AND Age = (6.5, 22.0> AND Fare = (52.277099609375, 59.
↪08749961853027> THEN Survived = {0}
IF Sex_female = (-inf, 0.5> AND Age = (77.0, inf) AND Fare = (-inf, 52.277099609375>␣
↪THEN Survived = {1}
IF Sex_female = (-inf, 0.5> AND Age = (-inf, 6.5> AND SibSp = (3.0, inf) AND Parch = (1.
↪5, inf) AND Fare = (-inf, 31.331250190734863> THEN Survived = {0}
IF Sex_female = (0.5, inf) AND Pclass_3 = (-inf, 0.5> AND Age = (-inf, 2.5> AND Pclass_2␣
↪= (-inf, 0.5> THEN Survived = {0}
```

3. Importance of features determined by RuleXAI

[25]:
```
explainer.feature_importances_
```

[25]:

	0 | attributes	0 | importances	1 | attributes	1 | importances
0	Pclass_3	0.895689	Sex_female	1.078293
1	Sex_female	0.844471	Age	0.973953
2	SibSp	0.683045	Fare	0.493457
3	Fare	0.584592	Pclass_3	0.140172
4	Age	0.396833	Pclass_2	0.126684
5	Pclass_2	0.186594	Embarked_C	0.11173
6	Embarked_C	0.101517	SibSp	0.089413
7	Parch	-0.051929	-	-
8	Pclass_1	-0.138521	-	-

Contrary to the importance of the features returned by the decision tree from sklearn, RuleXAI examines the importance of the conditions in the context of the class. In the case of this dataset, the importance of a feature tells us how much a given feature of a person contributed to the assignment of that person to this class. Analysing the ranking of the feature importance, it can be concluded that what most characterised the non-survivors was whether they traveled in grade 3 or not. Second important feature that had impact on non-survival was the gender of the person, and more precisely whether the person was a woman or not. We can also see that the number of siblings had an impact on survival - if someone had many siblings, the parents probably were not able to ensure the safety of all their children. Looking at the survivors, it can be seen that gender had the greatest impact on survival. We can also see that age had big impact on survival - we can draw the conclusions that probably at the beginning, children and the elderly were rescued, because people in their prime had a greater chance of surviving in difficult conditions. Next, whether someone survived depended on the fare - richer people were saved earlier than poorer.

From the feature ranking obtained with RuleXAI, similar conclusions can be drawn as with the feature ranking obtained with sklearn. The advantage of using RuleXAI, however, is that validity of the features is examined in the context of the class. Thanks to this, even without knowing the context, it would be known which feature influenced the assignment of a given class to a given object the most. An even more in-depth analysis can be performed using the ranking of conditions from the RuleXAI library, as will be shown later in the report

4. Model generation based on top 50% of features for each class from RuleXAI

[26]:
```
import numpy as np

features_importances_rulexai = explainer.feature_importances_

percent = 50
importances_TOP = []
for j in range(0, features_importances_rulexai.shape[1] + 0, 2):
    class_importances = (
        features_importances_rulexai.iloc[:, j]
        .replace("-", np.nan)
        .dropna()
    )
    class_importances_TOP_number = np.round(
        (percent / 100) * class_importances.shape[0]
    )

    if class_importances_TOP_number == 0:
        class_importances_TOP_number = 1

    class_importances_TOP = class_importances.loc[
        0: class_importances_TOP_number - 1
    ]
    importances_TOP.extend(list(class_importances_TOP))

importances_TOP_list = list(set(importances_TOP))

importances_TOP_list
```

[26]:
```
['Fare', 'Age', 'SibSp', 'Sex_female', 'Pclass_3']
```

[27]:
```
X_train_rulexai_features = X_train[importances_TOP_list]
X_test_rulexai_features = X_test[importances_TOP_list]

X_train_rulexai_features.head(5)
```

[27]:

	Fare	Age	SibSp	Sex_female	Pclass_3
445	81.8583	4.0	0	0	0
650	7.8958	28.0	0	0	1
172	11.1333	1.0	1	1	1
450	27.7500	36.0	1	0	0
314	26.2500	43.0	1	0	0

[28]:
```
dt_rulexai_features = DecisionTreeClassifier(random_state=1, max_depth=5)
cv = cross_val_score(dt_rulexai_features,
                     X_train_rulexai_features, y_train, cv=5, scoring = "balanced_
↪accuracy")
print(cv)
print(cv.mean())
```

```
[0.74440807 0.83791965 0.72110622 0.79403567 0.78065775]
0.7756254727056493
```

[29]:
```
dt_rulexai_features.fit(X_train_rulexai_features, y_train)
```

[29]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

[30]:
```
balanced_accuracy_score(y_test, dt_rulexai_features.predict(X_test_rulexai_features))
```

[30]:
```
0.8017157284673209
```

By selecting 50% of the most important features from the ranking obtained with RuleXAI, it can be seen that compared to the basic set, the results obtained by the decision tree have improved. Comparing these results with the results obtained for the set containing 50% of the most important features from the ranking determined with the use of sklearn, we can see that the results also have improved.

5. Further analysis using RuleXAI

5.1 Rule condition importance

Below we present how the importance of the conditions from rules derived from a decision tree can be analysed.

[31]:
```
explainer.condition_importances_
```

[31]:
```
                              0 | conditions_names  0 | importances  \
0                        Sex_female = (-inf, 0.5>         1.444185
1                           Pclass_3 = (0.5, inf)         0.856801
2                              SibSp = (3.0, inf)         0.683045
3   Fare = (149.035400390625, 152.5062484741211>         0.455265
4                                Age = (6.5, 22.0>         0.335094
5                Fare = (23.350000381469727, inf)         0.246165
6                               Age = (-inf, 2.5>         0.236955
7                               Age = (36.5, inf)         0.204974
8                          Pclass_2 = (-inf, 0.5>         0.186594
9              Fare = (-inf, 31.331250190734863>         0.119378
10               Fare = (-inf, 52.277099609375>         0.105219
11                     Embarked_C = (-inf, 0.5>         0.101517
12                             Age = (6.5, 77.0>         0.055288
13                          Parch = (-inf, 1.5>         0.053057
14                       Pclass_1 = (-inf, 0.5>         0.042272
15                       Pclass_3 = (-inf, 0.5>         0.038888
16             Fare = (-inf, 23.350000381469727>         0.038706
17                              Age = (7.0, inf)         0.032743
18                              Age = (6.5, inf)         0.030661
19                              Age = (2.5, inf)         0.005461
20                             Age = (-inf, 3.5>        -0.087280
21               Fare = (31.331250190734863, inf)        -0.103671
22                            Parch = (1.5, inf)        -0.104986
23                Fare = (59.08749961853027, inf)        -0.125345
24    Fare = (52.277099609375, 59.08749961853027>        -0.151125
25                          Pclass_1 = (0.5, inf)        -0.180793
26                             Age = (-inf, 6.5>        -0.417063
```

(continues on next page)

(continued from previous page)

```
27                        Sex_female = (0.5, inf)        -0.599715

                              1 | conditions_names 1 | importances
0                            Sex_female = (0.5, inf)        1.378759
1                                  Age = (77.0, inf)        0.404701
2                                  Age = (-inf, 6.5>        0.339353
3      Fare = (52.277099609375, 59.08749961853027>        0.285962
4                             Pclass_3 = (-inf, 0.5>        0.280239
5                                   Age = (3.5, 7.0>        0.254798
6                              Pclass_2 = (0.5, inf)        0.126684
7                      Fare = (59.08749961853027, inf)        0.120677
8                            Embarked_C = (0.5, inf)         0.11173
9                               SibSp = (-inf, 3.0>        0.089413
10  Fare = (15.372900009155273, 23.350000381469727>        0.084646
11                     Fare = (152.5062484741211, inf)        0.039821
12                    Fare = (23.350000381469727, inf)        0.033452
13                                 Age = (-inf, 2.5>        0.031559
14                                 Age = (22.0, inf)        0.011474
15                                Age = (-inf, 36.5>        0.011118
16                                  Age = (2.5, inf)       -0.003344
17                                 Age = (2.5, 49.5>       -0.006889
18                                  Age = (6.5, inf)       -0.008256
19                     Fare = (-inf, 149.035400390625>       -0.012183
20                      Fare = (-inf, 52.277099609375>       -0.027228
21                    Fare = (-inf, 15.372900009155273>       -0.031689
22                                 Age = (49.5, inf)       -0.060562
23                             Pclass_3 = (0.5, inf)       -0.140068
24                          Sex_female = (-inf, 0.5>       -0.300466
25                                                 -               -
26                                                 -               -
27                                                 -               -
```

Looking at the importance of individual conditions, we can see that the most important condition for a person not surviving is that the person was not a woman, that is, person was a man. On the other hand, the most important condition for a person to survive is that the person was a woman. In this way, we have an explicit confirmation of the hypothesis put forward on the basis of the data context during the analysis of the importance of the features. At this point, note that for categorical variables such as Sex_female, where the feature can be 0 or 1, the rules taken from the decision tree return the condition Sex_female = (-inf, 0.5> when the feature is 0 and Sex_female = (0.5, inf) when the feature takes the value 1.

We also see that the second condition determing that the person did not survive is that they traveled in 3rd grade. This also confirms the hypothesis put forward during the analysis of the feature importance. In the case of the feature importance analysis, we only had information on whether survival was affected by the fact that the person was travelling in 3rd grade. The ranking of conditions gives us an unambiguous confirmation of which decision is impacted by this feature.

The situation is similar with the number of siblings. We can see that if a person had more than 3 siblings, it was more likely that they were in the group of non-survivors. This confirms the hypothesis put forward earlier: with more children, the parents were not able to ensure the safety of all their children.

On the other hand, when looking at the conditions for survivors, it can be seen that in addition to being a woman, the following conditions rank high: Age = (77.0, inf), Age = (-inf, 6.5>. This confirms the hypothesis that children and the elderly people were saved first.

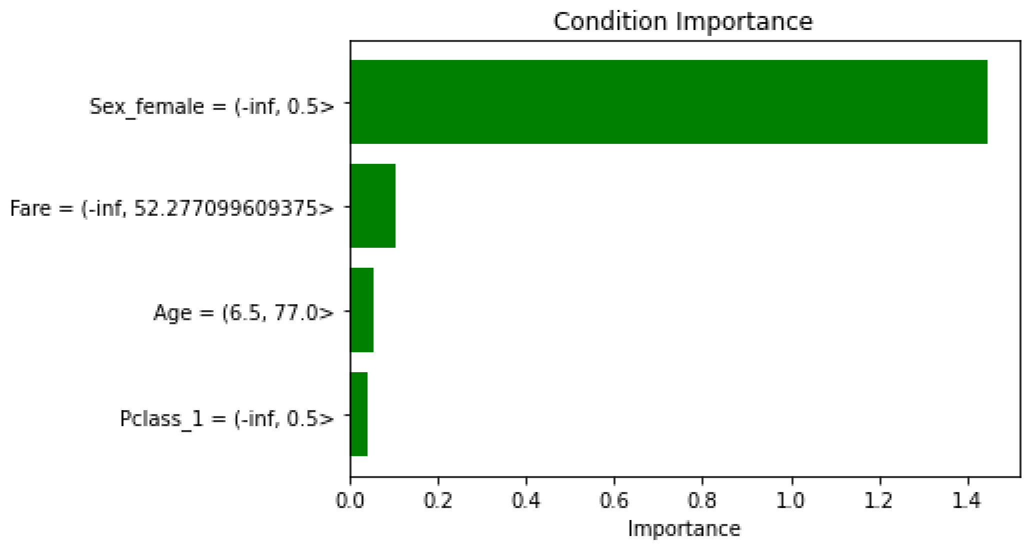
5.2 Local explainability

It is often interesting and important to know on what criteria the model made its decision for a given example. In general, thanks to the explanations obtained with XAI methods, the correctness of the model can be verified. Additionally, in some applications of AI it is important that thanks to the use of XAI people affected by the model's decision better understand their situation and have more trust in the model.

This type of explanation is provided by RuleXAI. The explanations take the form of easy to understand and interpret rules, based on which the model makes a decision for a given example, and the importance of the conditions contained in them.

[32]:
```
example_X = X_train.iloc[1, :]
example_Y = pd.DataFrame(y_train).iloc[1, :]

explainer.local_explainability(example_X, example_Y, plot = True)
```

```
Example:
Age            28.0
SibSp           0.0
Parch           0.0
Fare          7.8958
Pclass_1        0.0
Pclass_2        0.0
Pclass_3        1.0
Sex_female      0.0
Sex_male        1.0
Embarked_C      0.0
Embarked_Q      0.0
Embarked_S      1.0
Survived          0
Name: 650, dtype: object

Rules that covers this example:
IF Sex_female = (-inf, 0.5> AND Age = (6.5, 77.0> AND Fare = (-inf, 52.277099609375> AND␣
↪Pclass_1 = (-inf, 0.5> THEN Survived = {0}

Importances of the conditions from rules covering the example
             0 | conditions_names  0 | importances
0        Sex_female = (-inf, 0.5>         1.444185
1  Fare = (-inf, 52.277099609375>         0.105219
2               Age = (6.5, 77.0>         0.055288
3          Pclass_1 = (-inf, 0.5>         0.042272
```



[32]:
```
               0 | conditions_names  0 | importances
0        Sex_female = (-inf, 0.5>         1.444185
1  Fare = (-inf, 52.277099609375>         0.105219
2               Age = (6.5, 77.0>         0.055288
3          Pclass_1 = (-inf, 0.5>         0.042272
```

Looking at the explanation, we can see that the model classifies a person as a non-survivor based on the rule stating that the person not survived because: was male, was beetwen (6.5, 77.0> years old, paid little for the fare and did not travel in 1st class. The most important of these conditions was that this person was male.

5.3 Creation of a binary dataset

Another functionality provided by RuleXAI is the conversion of the input dataset into a set described by binary features that correspond to specific conditions determined by the model. If for a given condition the example takes the value 0, it means that it does not meet it. If, on the other hand, it takes 1, it means that it satisfies it. This dataset can be used to train other ML models. A significant advantage of such a set is that it has no missing values and only has one type of data (it can be considered as categorical or numerical). Thanks to this, it can be used with any of the available ML models.

[33]:
```
X_train_tranformed = explainer.fit_transform(X_train, selector=None)

X_train_tranformed.head(5)
```

[33]:
```
  Sex_female = (-inf, 0.5> Age = (6.5, 77.0> Fare = (-inf, 52.277099609375>  \
0                        1                 0                              0
1                        1                 1                              1
2                        0                 0                              1
3                        1                 1                              1
4                        1                 1                              1

  Pclass_1 = (-inf, 0.5> Sex_female = (0.5, inf) Pclass_3 = (-inf, 0.5>  \
0                      0                       0                      1
1                      1                       0                      0
2                      1                       1                      0
```

(continues on next page)

(continued from previous page)

```
3                       1                        0                       1
4                       1                        0                       1

  Age = (2.5, 49.5> Fare = (-inf, 149.035400390625> Pclass_3 = (0.5, inf)  \
0                  1                               1                     0
1                  1                               1                     1
2                  0                               1                     1
3                  1                               1                     0
4                  1                               1                     0

  Fare = (-inf, 15.372900009155273>  ... Parch = (1.5, inf)  \
0                                 0  ...                  1
1                                 1  ...                  0
2                                 1  ...                  0
3                                 0  ...                  1
4                                 0  ...                  0

  Fare = (31.331250190734863, inf) Age = (-inf, 2.5> Pclass_2 = (0.5, inf)  \
0                                1                 0                     0
1                                0                 0                     0
2                                0                 1                     0
3                                0                 0                     1
4                                0                 0                     1

  Age = (3.5, 7.0> Fare = (149.035400390625, 152.5062484741211>  \
0                 1                                             0
1                 0                                             0
2                 0                                             0
3                 0                                             0
4                 0                                             0

  Age = (6.5, 22.0> Age = (77.0, inf) Fare = (-inf, 31.331250190734863>  \
0                  0                 0                                  0
1                  0                 0                                  1
2                  0                 0                                  1
3                  0                 0                                  1
4                  0                 0                                  1

  Pclass_2 = (-inf, 0.5>
0                       1
1                       1
2                       1
3                       0
4                       0

[5 rows x 41 columns]
```

[37]:
```
X_test_transformed = explainer.transform(X_test)

X_test_transformed.head(5)
```

[37]:

```
  Sex_female = (-inf, 0.5> Age = (6.5, 77.0> Fare = (-inf, 52.277099609375>  \
0                        1                 1                                1
1                        1                 1                                1
2                        1                 1                                1
3                        0                 0                                1
4                        0                 1                                1

  Pclass_1 = (-inf, 0.5> Sex_female = (0.5, inf) Pclass_3 = (-inf, 0.5>  \
0                      1                       0                      0
1                      1                       0                      1
2                      1                       0                      0
3                      1                       1                      1
4                      1                       1                      0

  Age = (2.5, 49.5> Fare = (-inf, 149.035400390625> Pclass_3 = (0.5, inf)  \
0                 1                               1                     1
1                 1                               1                     0
2                 1                               1                     1
3                 1                               1                     0
4                 1                               1                     1

  Fare = (-inf, 15.372900009155273>  ... Parch = (1.5, inf)  \
0                                 1  ...                  0
1                                 1  ...                  0
2                                 1  ...                  0
3                                 0  ...                  0
4                                 1  ...                  0

  Fare = (31.331250190734863, inf) Age = (-inf, 2.5> Pclass_2 = (0.5, inf)  \
0                                0                 0                     0
1                                0                 0                     1
2                                0                 0                     0
3                                1                 0                     1
4                                0                 0                     0

  Age = (3.5, 7.0> Fare = (149.035400390625, 152.5062484741211>  \
0                0                                             0
1                0                                             0
2                0                                             0
3                1                                             0
4                0                                             0

  Age = (6.5, 22.0> Age = (77.0, inf) Fare = (-inf, 31.331250190734863>  \
0                 0                 0                                 1
1                 0                 0                                 1
2                 1                 0                                 1
3                 0                 0                                 0
4                 1                 0                                 1

  Pclass_2 = (-inf, 0.5>
0                      1
1                      0
2                      1
```

(continues on next page)

(continued from previous page)

```
3                       0
4                       1

[5 rows x 41 columns]
```

[35]:
```
dt_binary_dataset = DecisionTreeClassifier(random_state=1, max_depth=5)
cv = cross_val_score(dt_binary_dataset,
                     X_train_tranformed, y_train, cv=5, scoring = "balanced_accuracy")
print(cv)
print(cv.mean())
```

```
[0.76991271 0.83970831 0.74009356 0.7990524  0.84448161]
0.7986497169696036
```

[36]:
```
dt_binary_dataset.fit(X_train_tranformed, y_train)
```

[36]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

[38]:
```
balanced_accuracy_score(y_test, dt_binary_dataset.predict(X_test_transformed))
```

[38]:
```
0.7807425259654559
```

Looking at the results obtained by the decision tree trained on the created binary set, we can see that it obtains very similar (even slightly better) results than on the original dataset.

5.4 Creation of a binary dataset based on top conditions

RuleXAI allows you to create a binary dataset with a selected percentage of the most important conditions.

[39]:
```
X_train_tranformed = explainer.fit_transform(X_train, selector=0.25)

X_train_tranformed.head(5)
```

[39]:
```
  SibSp = (3.0, inf) Fare = (52.277099609375, 59.08749961853027>  \
0                  0                                            0
1                  0                                            0
2                  0                                            0
3                  0                                            0
4                  0                                            0

  Pclass_3 = (-inf, 0.5> Sex_female = (-inf, 0.5> Sex_female = (0.5, inf)  \
0                      1                        1                        0
1                      0                        1                        0
2                      0                        0                        1
3                      1                        1                        0
4                      1                        1                        0

  Age = (3.5, 7.0> Fare = (149.035400390625, 152.5062484741211>  \
0                1                                            0
1                0                                            0
2                0                                            0
3                0                                            0
```

(continues on next page)

(continued from previous page)

```
4              0                                          0

  Age = (6.5, 22.0> Age = (-inf, 2.5> Age = (-inf, 6.5> Pclass_3 = (0.5, inf)  \
0                 0                 0                 1                     0
1                 0                 0                 0                     1
2                 0                 1                 1                     1
3                 0                 0                 0                     0
4                 0                 0                 0                     0

  Fare = (23.350000381469727, inf) Age = (77.0, inf)
0                                1                 0
1                                0                 0
2                                0                 0
3                                1                 0
4                                1                 0
```

[44]:
```
X_test_transformed = explainer.transform(X_test)

X_test_transformed.head(5)
```

[44]:
```
  SibSp = (3.0, inf) Fare = (52.277099609375, 59.08749961853027>  \
0                  0                                           0
1                  0                                           0
2                  0                                           0
3                  0                                           0
4                  0                                           0

  Pclass_3 = (-inf, 0.5> Sex_female = (-inf, 0.5> Sex_female = (0.5, inf)  \
0                      0                        1                       0
1                      1                        1                       0
2                      0                        1                       0
3                      1                        0                       1
4                      0                        0                       1

  Age = (3.5, 7.0> Fare = (149.035400390625, 152.5062484741211>  \
0                0                                            0
1                0                                            0
2                0                                            0
3                1                                            0
4                0                                            0

  Age = (6.5, 22.0> Age = (-inf, 2.5> Age = (-inf, 6.5> Pclass_3 = (0.5, inf)  \
0                 0                 0                 0                     1
1                 0                 0                 0                     0
2                 1                 0                 0                     1
3                 0                 0                 1                     0
4                 1                 0                 0                     1

  Fare = (23.350000381469727, inf) Age = (77.0, inf)
0                                0                 0
1                                0                 0
2                                0                 0
```

(continues on next page)

(continued from previous page)

```
3                                1                 0
4                                0                 0
```

[41]:
```
dt_binary_dataset_with_TOP_conditions = DecisionTreeClassifier(random_state=1, max_
↪depth=5)
cv = cross_val_score(dt_binary_dataset_with_TOP_conditions,
                     X_train_tranformed, y_train, cv=5, scoring = "balanced_accuracy")
print(cv)
print(cv.mean())
```

```
[0.77414075 0.86419923 0.76004403 0.79849498 0.81633222]
0.8026422425531315
```

[42]:
```
dt_binary_dataset_with_TOP_conditions.fit(X_train_tranformed,y_train)
```

[42]:
```
DecisionTreeClassifier(max_depth=5, random_state=1)
```

[45]:
```
balanced_accuracy_score(y_test, dt_binary_dataset_with_TOP_conditions.predict(X_test_
↪transformed))
```

[45]:
```
0.7701841969357893
```

The results obtained on a binary set containing only 25% of all conditions are very similar to those obtained on the entire set. We can see that with fewer data dimensions, similar results can be obtained. Under appropriate conditions, a binary set can be used also to reduce the dimensionality of the set.

5.5 Condition importance based on non-overlapping rule conditions

Below we present another analysis of the importance of the conditions from rules derived from a decision tree. This time the analysis focuses on conditions in rules splitted into base conditions so that individual conditions do not overlap.

[47]:
```
explainer.explain(basic_conditions=True)
explainer.condition_importances_
```

[47]:
```
                                 0 | conditions_names  0 | importances  \
0                        Sex_female = (-inf, 0.5)         0.709511
1                              SibSp = <3.0, inf)         0.365199
2                            Pclass_3 = <0.5, inf)        0.359394
3      Fare = <149.035400390625, 152.5062484741211)       0.345837
4                              Age = <7.0, 22.0)          0.205906
5                  Fare = (-inf, 15.372900009155273)      0.201038
6                              Age = (-inf, 2.5)          0.188995
7                          Pclass_2 = (-inf, 0.5)         0.186594
8   Fare = <23.350000381469727, 31.331250190734863)       0.094355
9                          Pclass_3 = (-inf, 0.5)         0.086410
10                            Age = <36.5, 49.5)          0.083850
11                            Age = <22.0, 36.5)          0.074990
12     Fare = <31.331250190734863, 52.277099609375)       0.045709
13                       Embarked_C = (-inf, 0.5)         0.036915
14                          Parch = (-inf, 1.5)           0.035371
15                            Age = <49.5, 77.0)          0.031508
16                         Pclass_1 = (-inf, 0.5)         0.015372
```

(continues on next page)

(continued from previous page)

```
17                                 Age = <6.5, 7.0)        0.000000
18                               Parch = <1.5, inf)       -0.049627
19                            Pclass_1 = <0.5, inf)       -0.065743
20  Fare = <15.372900009155273, 23.350000381469727)       -0.104907
21     Fare = <59.08749961853027, 149.035400390625)       -0.137746
22                 Fare = <152.5062484741211, inf)        -0.191079
23                                 Age = <3.5, 6.5)       -0.222782
24      Fare = <52.277099609375, 59.08749961853027)       -0.266055
25                          Sex_female = <0.5, inf)       -0.330555
26                                Age = <77.0, inf)       -0.340750
27                                 Age = <2.5, 3.5)       -0.411382

                           1 | conditions_names 1 | importances
0                           Sex_female = <0.5, inf)       0.593139
1                                 Age = <77.0, inf)       0.427564
2                                  Age = <3.5, 6.5)       0.306936
3       Fare = <52.277099609375, 59.08749961853027)       0.243723
4                                  Age = <2.5, 3.5)       0.217869
5                           Pclass_3 = (-inf, 0.5)        0.129432
6                                 Age = (-inf, 2.5)       0.127256
7                             Pclass_2 = <0.5, inf)       0.126684
8      Fare = <59.08749961853027, 149.035400390625)       0.092037
9   Fare = <15.372900009155273, 23.350000381469727)        0.09141
10                 Fare = <152.5062484741211, inf)        0.079653
11  Fare = <23.350000381469727, 31.331250190734863)       0.075764
12                            SibSp = (-inf, 3.0)         0.053725
13                                Age = <22.0, 36.5)      0.051714
14                          Embarked_C = <0.5, inf)       0.040629
15                                Age = <49.5, 77.0)      0.026622
16     Fare = <31.331250190734863, 52.277099609375)       0.025254
17                                 Age = <6.5, 7.0)            0.0
18                                Age = <36.5, 49.5)     -0.001405
19                                 Age = <7.0, 22.0)     -0.035978
20               Fare = (-inf, 15.372900009155273)       -0.052213
21     Fare = <149.035400390625, 152.5062484741211)       -0.05625
22                            Pclass_3 = <0.5, inf)      -0.060642
23                         Sex_female = (-inf, 0.5)      -0.151878
24                                                -              -
25                                                -              -
26                                                -              -
27                                                -              -
```

Looking at the ranking of conditions that do not overlap, we can come to conclusions similar to the ones drawn from the ranking of overlaping conditions. However, differences can also be seen, e.g. the condition Age = (-inf, 6.5> was high in the ranking of overlapping conditions, suggesting the conclusion that all children under 6.5 were more likely to survive. Nevertheless, when we look at the assessment of the basic conditions, we can see that children aged between 2.5 and 6.5 years [Age = <2.5, 3.5), Age = <3.5, 6.5)] had a good chance of survival, and the condition for children under 2.5 is lower in the ranking. This suggests that infants (under 2.5 years of age) were less likely to survive, possibly due to their dependency.

5.3 Presentation of the rules consisting of non-overlapping base conditions

[48]:
```python
rules = explainer.get_rules_with_basic_conditions()
for rule in rules:
    print(rule)
```

```
IF [Sex_female = (-inf, 0.5)] AND [Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0,␣
↪36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.0)] AND [Fare = (-inf, 15.
↪372900009155273) OR Fare = <15.372900009155273, 23.350000381469727) OR Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)]␣
↪AND [Pclass_1 = (-inf, 0.5)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = <2.5, 3.5) OR Age = <3.5, 6.5) OR Age = <6.5, 7.
↪0) OR Age = <7.0, 22.0) OR Age = <22.0, 36.5) OR Age = <36.5, 49.5)] AND [Fare = (-inf,
↪ 15.372900009155273) OR Fare = <15.372900009155273, 23.350000381469727) OR Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)␣
↪OR Fare = <52.277099609375, 59.08749961853027) OR Fare = <59.08749961853027, 149.
↪035400390625)] AND [Pclass_3 = (-inf, 0.5)] THEN Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5, 6.
↪5) OR Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0, 36.5)] AND [Fare = (-inf,␣
↪15.372900009155273)] AND [Pclass_3 = <0.5, inf)] THEN Survived = {1}
IF [Sex_female = (-inf, 0.5)] AND [Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0,␣
↪36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.0)] AND [Fare = (-inf, 15.
↪372900009155273) OR Fare = <15.372900009155273, 23.350000381469727) OR Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)]␣
↪AND [Pclass_1 = <0.5, inf)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5, 6.
↪5) OR Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0, 36.5)] AND [Fare = <15.
↪372900009155273, 23.350000381469727)] AND [Pclass_3 = <0.5, inf)] THEN Survived = {1}
IF [Sex_female = (-inf, 0.5)] AND [Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0,␣
↪36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.0) OR Age = <77.0, inf)] AND [Fare =
↪<59.08749961853027, 149.035400390625) OR Fare = <149.035400390625, 152.5062484741211)␣
↪OR Fare = <152.5062484741211, inf)] AND [Embarked_C = (-inf, 0.5)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = <7.0, 22.0) OR Age = <22.0, 36.5) OR Age = <36.5,
↪ 49.5) OR Age = <49.5, 77.0) OR Age = <77.0, inf)] AND [Fare = <23.350000381469727, 31.
↪331250190734863) OR Fare = <31.331250190734863, 52.277099609375) OR Fare = <52.
↪277099609375, 59.08749961853027) OR Fare = <59.08749961853027, 149.035400390625) OR␣
↪Fare = <149.035400390625, 152.5062484741211) OR Fare = <152.5062484741211, inf)] AND␣
↪[Pclass_3 = <0.5, inf)] THEN Survived = {0}
IF [Sex_female = (-inf, 0.5)] AND [Age = <6.5, 7.0) OR Age = <7.0, 22.0) OR Age = <22.0,␣
↪36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.0) OR Age = <77.0, inf)] AND [Fare =
↪<59.08749961853027, 149.035400390625) OR Fare = <149.035400390625, 152.5062484741211)␣
↪OR Fare = <152.5062484741211, inf)] AND [Embarked_C = <0.5, inf)] THEN Survived = {1}
IF [Sex_female = (-inf, 0.5)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5,␣
↪6.5)] AND [SibSp = (-inf, 3.0)] THEN Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = <49.5, 77.0) OR Age = <77.0, inf)] AND [Fare = (-
↪inf, 15.372900009155273) OR Fare = <15.372900009155273, 23.350000381469727) OR Fare =
↪<23.350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.
↪277099609375) OR Fare = <52.277099609375, 59.08749961853027) OR Fare = <59.
↪08749961853027, 149.035400390625)] AND [Pclass_3 = (-inf, 0.5)] THEN Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = <2.5, 3.5) OR Age = <3.5, 6.5) OR Age = <6.5, 7.
↪0) OR Age = <7.0, 22.0) OR Age = <22.0, 36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.
↪0) OR Age = <77.0, inf)] AND [Fare = <152.5062484741211, inf)] AND [Pclass_3 = (-inf,␣
↪0.5)] THEN Survived = {1}
```

(continues on next page)

(continued from previous page)

```
IF [Sex_female = (-inf, 0.5)] AND [Age = <22.0, 36.5) OR Age = <36.5, 49.5) OR Age = <49.
↪5, 77.0) OR Age = <77.0, inf)] AND [Fare = <52.277099609375, 59.08749961853027)] THEN␣
↪Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = <36.5, 49.5) OR Age = <49.5, 77.0) OR Age = <77.
↪0, inf)] AND [Fare = (-inf, 15.372900009155273) OR Fare = <15.372900009155273, 23.
↪350000381469727)] AND [Pclass_3 = <0.5, inf)] THEN Survived = {0}
IF [Sex_female = (-inf, 0.5)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5,␣
↪6.5)] AND [SibSp = <3.0, inf)] AND [Parch = (-inf, 1.5)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5)] AND [Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)␣
↪OR Fare = <52.277099609375, 59.08749961853027) OR Fare = <59.08749961853027, 149.
↪035400390625) OR Fare = <149.035400390625, 152.5062484741211) OR Fare = <152.
↪5062484741211, inf)] AND [Pclass_3 = <0.5, inf)] THEN Survived = {0}
IF [Sex_female = (-inf, 0.5)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5,␣
↪6.5)] AND [Fare = <31.331250190734863, 52.277099609375) OR Fare = <52.277099609375, 59.
↪08749961853027) OR Fare = <59.08749961853027, 149.035400390625) OR Fare = <149.
↪035400390625, 152.5062484741211) OR Fare = <152.5062484741211, inf)] AND [SibSp = <3.0,
↪ inf)] AND [Parch = <1.5, inf)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = (-inf, 2.5)] AND [Pclass_3 = (-inf, 0.5)] AND␣
↪[Pclass_2 = <0.5, inf)] THEN Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = <3.5, 6.5) OR Age = <6.5, 7.0)] AND [Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)␣
↪OR Fare = <52.277099609375, 59.08749961853027) OR Fare = <59.08749961853027, 149.
↪035400390625) OR Fare = <149.035400390625, 152.5062484741211) OR Fare = <152.
↪5062484741211, inf)] AND [Pclass_3 = <0.5, inf)] THEN Survived = {1}
IF [Sex_female = <0.5, inf)] AND [Age = <2.5, 3.5) OR Age = <3.5, 6.5) OR Age = <6.5, 7.
↪0) OR Age = <7.0, 22.0) OR Age = <22.0, 36.5) OR Age = <36.5, 49.5) OR Age = <49.5, 77.
↪0) OR Age = <77.0, inf)] AND [Fare = <149.035400390625, 152.5062484741211)] AND␣
↪[Pclass_3 = (-inf, 0.5)] THEN Survived = {0}
IF [Sex_female = (-inf, 0.5)] AND [Age = <6.5, 7.0) OR Age = <7.0, 22.0)] AND [Fare =
↪<52.277099609375, 59.08749961853027)] THEN Survived = {0}
IF [Sex_female = (-inf, 0.5)] AND [Age = <77.0, inf)] AND [Fare = (-inf, 15.
↪372900009155273) OR Fare = <15.372900009155273, 23.350000381469727) OR Fare = <23.
↪350000381469727, 31.331250190734863) OR Fare = <31.331250190734863, 52.277099609375)]␣
↪THEN Survived = {1}
IF [Sex_female = (-inf, 0.5)] AND [Age = (-inf, 2.5) OR Age = <2.5, 3.5) OR Age = <3.5,␣
↪6.5)] AND [Fare = (-inf, 15.372900009155273) OR Fare = <15.372900009155273, 23.
↪350000381469727) OR Fare = <23.350000381469727, 31.331250190734863)] AND [SibSp = <3.0,
↪ inf)] AND [Parch = <1.5, inf)] THEN Survived = {0}
IF [Sex_female = <0.5, inf)] AND [Age = (-inf, 2.5)] AND [Pclass_3 = (-inf, 0.5)] AND␣
↪[Pclass_2 = (-inf, 0.5)] THEN Survived = {0}
```

IV. Using the RuleKit library - a versatile tool for rule learning - to generate rules

In the previous section, the rules obtained from the decision tree were analysed. In this section the analysis is based on rules obtained using the algorithm dedicated for rule-based learning. A set of such algorithms is provided by the RuleKit library.

1. Data preparation for RuleKit

RuleKit supports missing values and categorical data, so the step of preparing data specifically for this algorithm is not necessary - what was done in Section I is sufficient. The only thing to remember is that RuleKit accepts missing values for numeric columns as nan, while for categorical columns as None. For this reason it was necessary to change the missing values in the Embarked column from nan to None.

[49]:
```python
import numpy as np

X = data.drop(["Survived"], axis=1)
X.Embarked.replace(np.nan, None, inplace = True)
y = data.Survived

X.head(5)
```

[49]:
```
   Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0       3    male  22.0      1      0   7.2500        S
1       1  female  38.0      1      0  71.2833        C
2       3  female  26.0      0      0   7.9250        S
3       1  female  35.0      1      0  53.1000        S
4       3    male  35.0      0      0   8.0500        S
```

2. Data split for training and test datasets

[50]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

3. Building and testing the model

[51]:
```python
from sklearn.model_selection import cross_val_score
from rulekit import RuleKit
from rulekit.classification import RuleClassifier
from rulekit.params import Measures

RuleKit.init()

rc = RuleClassifier(
    induction_measure=Measures.C2,
    pruning_measure=Measures.C2,
    voting_measure=Measures.C2,
    min_rule_covered=5
```

(continues on next page)

(continued from previous page)

```
)
cv = cross_val_score(rc, X_train, y_train, cv=5, scoring = "balanced_accuracy")
print(cv)
print(cv.mean())
```

```
[0.73958333 0.88666667 0.70763889 0.8027027  0.82383191]
0.7920847009219034
```

[52]:
```
rc.fit(X_train, y_train)
```

[52]:
```
<rulekit.classification.RuleClassifier at 0x29d6da16fd0>
```

[53]:
```
balanced_accuracy_score(y_test, rc.predict(X_test))
```

[53]:
```
0.749756125552304
```

The quality of the rule model is similar to that obtained for the decision tree. By controlling the rule model induction parameters even better results could be obtained, but this is not the subject of this notebook.

4. Presentation of the rules obtained by RuleKit

[54]:
```
for rule in rc.model.rules:
    print(rule)
```

```
IF Age = (-inf, 0.92) THEN Survived = {1}
IF Pclass = {2} AND Sex = {female} THEN Survived = {1}
IF Parch = (-inf, 1.50) AND Sex = {female} AND Fare = <29.36, inf) THEN Survived = {1}
IF Pclass = {1} AND Sex = {female} AND Age = <8, inf) AND Fare = <29.36, inf) THEN␣
↪Survived = {1}
IF Parch = (-inf, 1.50) AND Sex = {female} AND SibSp = (-inf, 2.50) AND Fare = <21.72,␣
↪inf) THEN Survived = {1}
IF Sex = {female} AND SibSp = <0.50, inf) AND Age = <27.50, inf) AND Fare = <15.98, 22.
↪34) THEN Survived = {1}
IF Parch = (-inf, 1.50) AND Sex = {female} AND SibSp = (-inf, 2.50) AND Fare = <10.48,␣
↪inf) THEN Survived = {1}
IF Parch = (-inf, 3.50) AND Sex = {female} AND SibSp = (-inf, 0.50) AND Fare = <10.83,␣
↪inf) THEN Survived = {1}
IF Parch = (-inf, 1.50) AND Sex = {female} AND Fare = <6.99, inf) THEN Survived = {1}
IF SibSp = (-inf, 2) AND Age = <8, 43.50) AND Fare = <82.66, inf) THEN Survived = {1}
IF SibSp = (-inf, 1.50) AND Age = <3, 62) AND Fare = <74.38, inf) THEN Survived = {1}
IF SibSp = (-inf, 1.50) AND Age = <22.50, 44.50) AND Fare = <52.28, 143.59) THEN␣
↪Survived = {1}
IF Pclass = {2} AND Age = (-inf, 6.50) THEN Survived = {1}
IF Embarked = {S} AND Age = <2.50, 6.50) THEN Survived = {1}
IF Parch = (-inf, 0.50) AND Fare = <29.85, inf) THEN Survived = {1}
IF Pclass = {1} AND Parch = (-inf, 0.50) AND Age = (-inf, 48.50) AND Fare = <26.14, 30.
↪75) THEN Survived = {1}
IF Embarked = {C} AND Age = (-inf, 29.50) AND Fare = <7.56, 135.07) THEN Survived = {1}
IF Parch = (-inf, 3.50) AND SibSp = (-inf, 2.50) AND Age = (-inf, 51.50) AND Fare = <18.
↪38, inf) THEN Survived = {1}
IF Age = (-inf, 42.50) AND Fare = <10.48, inf) THEN Survived = {1}
```

(continues on next page)

(continued from previous page)

```
IF SibSp = (-inf, 0.50) AND Age = <30.50, 44.50) AND Fare = <7.91, inf) THEN Survived =
↪{1}
IF Parch = (-inf, 3.50) AND SibSp = (-inf, 2.50) AND Fare = <7.91, inf) THEN Survived =
↪{1}
IF Age = <20, 27.50) AND Fare = <7.13, 7.80) THEN Survived = {1}
IF Parch = (-inf, 3.50) AND SibSp = (-inf, 4.50) AND Age = (-inf, 60.50) AND Fare = <7.
↪69, inf) THEN Survived = {1}
IF SibSp = <4.50, inf) THEN Survived = {0}
IF Parch = <3.50, inf) THEN Survived = {0}
IF Age = <26.50, inf) AND Fare = (-inf, 7.13) THEN Survived = {0}
IF Fare = <2.01, 7.13) THEN Survived = {0}
IF Sex = {male} AND Age = <13, 60.50) AND Fare = (-inf, 26.27) THEN Survived = {0}
IF Sex = {male} AND Age = <13, 77) AND Fare = (-inf, 52.28) THEN Survived = {0}
IF Embarked = {S} AND Sex = {male} AND Age = <13, 30.50) AND Fare = <7.80, inf) THEN␣
↪Survived = {0}
IF Sex = {male} AND Age = <13, 77) AND Fare = (-inf, 86.29) THEN Survived = {0}
IF Sex = {male} AND Age = <13, 77) AND Fare = (-inf, 387.66) THEN Survived = {0}
IF Pclass = {3} AND Sex = {male} AND Age = <6.50, 25.50) AND Fare = <16, inf) THEN␣
↪Survived = {0}
IF Pclass = {3} AND Fare = <23.35, 52.28) THEN Survived = {0}
IF Fare = <9.41, 10.48) THEN Survived = {0}
IF Embarked = {S} AND Age = <20.50, 30.50) AND Fare = <7.87, 8.08) THEN Survived = {0}
IF Embarked = {S} AND Parch = (-inf, 0.50) AND Fare = (-inf, 10.48) THEN Survived = {0}
IF Age = <17.50, inf) AND Fare = (-inf, 10.48) THEN Survived = {0}
IF Embarked = {S} AND Pclass = {3} AND Parch = (-inf, 0.50) AND SibSp = (-inf, 2.50) AND␣
↪Age = <16.50, inf) AND Fare = <7.99, 51.70) THEN Survived = {0}
IF Pclass = {3} AND Age = <6.50, inf) AND Fare = <13.29, 15.17) THEN Survived = {0}
IF Parch = (-inf, 2.50) AND Age = <2.50, inf) AND Fare = (-inf, 29.41) THEN Survived =
↪{0}
```

V. Analysis with RuleXAI of rules derived with RuleKit

1. Initialisation and explaination

[55]:
```python
from rulexai.explainer import RuleExplainer
explainer = RuleExplainer(model=rc, X=X_train, y=y_train, type="classification")
explainer.explain()
```

[55]:
```
<rulexai.explainer.RuleExplainer at 0x29d6e38f3a0>
```

2. Feature importance determined by RuleXAI

[56]:
```python
explainer.feature_importances_
```

[56]:
```
  1 | attributes  1 | importances 0 | attributes  0 | importances
0            Sex         2.153698           Fare         2.745895
1           Fare         1.834094            Sex         1.481201
2            Age         1.518890         Pclass         0.731226
3         Pclass         0.437313          Parch         0.571222
```

(continues on next page)

(continued from previous page)

```
4          SibSp         0.161694          Age          0.550799
5       Embarked         0.135499        SibSp          0.513230
6          Parch         0.089201     Embarked          0.205367
```

3. Rule condition importance

[57]:
```
explainer.condition_importances_
```

[57]:
```
       1 | conditions_names  1 | importances   0 | conditions_names  \
0          Sex = {female}          2.153698           Sex = {male}
1        Age = (-inf, 0.92)        0.510823          Pclass = {3}
2          Age = <2.5, 6.5)        0.426003     Fare = <2.01, 7.13)
3         Age = (-inf, 6.5)        0.359902    Fare = <9.41, 10.48)
4              Pclass = {2}        0.275112   Fare = (-inf, 10.48)
5       Fare = <82.66, inf)        0.267579      SibSp = <4.5, inf)
6    Fare = <52.28, 143.59)        0.257940      Parch = <3.5, inf)
7       Fare = <29.85, inf)        0.247359    Fare = (-inf, 7.13)
8       Fare = <74.38, inf)        0.239459         Embarked = {S}
9       Fare = <10.48, inf)        0.213492     Fare = <7.87, 8.08)
10             Pclass = {1}        0.162201     Age = <13.0, 77.0)
11           Embarked = {C}        0.154946   Fare = <13.29, 15.17)
12      Fare = <29.36, inf)        0.149020   Fare = <23.35, 52.28)
13       Fare = <7.91, inf)        0.125088   Fare = (-inf, 29.41)
14      Fare = <18.38, inf)        0.098349     Age = <20.5, 30.5)
15      Parch = (-inf, 1.5)        0.071067   Fare = (-inf, 26.27)
16      SibSp = (-inf, 1.5)        0.066773   Fare = (-inf, 52.28)
17      Fare = <21.72, inf)        0.065569      Age = <6.5, 25.5)
18      Fare = <10.83, inf)        0.062618    Parch = (-inf, 0.5)
19       Age = <20.0, 27.5)        0.059567     Age = <13.0, 30.5)
20       Age = <30.5, 44.5)        0.057521        Age = <6.5, inf)
21      SibSp = (-inf, 2.5)        0.056610       Age = <17.5, inf)
22        Age = <27.5, inf)        0.043101     Age = <13.0, 60.5)
23   Fare = <15.98, 22.34)         0.035205       Age = <26.5, inf)
24   Fare = <7.56, 135.07)         0.032317   Fare = (-inf, 86.29)
25       Age = (-inf, 48.5)        0.030201        Age = <2.5, inf)
26       SibSp = <0.5, inf)        0.028345    Fare = <7.99, 51.7)
27   Fare = <26.14, 30.75)         0.026043       Age = <16.5, inf)
28       Age = <22.5, 44.5)        0.024332  Fare = (-inf, 387.66)
29      Parch = (-inf, 3.5)        0.024229    Parch = (-inf, 2.5)
30       Fare = <7.69, inf)        0.018042    SibSp = (-inf, 2.5)
31      SibSp = (-inf, 2.0)        0.008422        Fare = <7.8, inf)
32       Fare = <6.99, inf)        0.007874      Fare = <16.0, inf)
33      SibSp = (-inf, 4.5)        0.006384                       -
34        Age = <3.0, 62.0)        0.004901                       -
35        Age = <8.0, 43.5)        0.004844                       -
36      Age = (-inf, 51.5)         0.002246                       -
37      Age = (-inf, 60.5)         0.002245                       -
38      Age = (-inf, 42.5)         0.000703                       -
39         Age = <8.0, inf)       -0.003570                       -
40      Age = (-inf, 29.5)        -0.003928                       -
41      SibSp = (-inf, 0.5)       -0.004841                       -
```

(continues on next page)

(continued from previous page)

```
42     Parch = (-inf, 0.5)        -0.006095                     -
43      Fare = <7.13, 7.8)        -0.011860                     -
44          Embarked = {S}        -0.019447                     -

   0 | importances
0         1.481201
1         0.731226
2         0.519133
3         0.517857
4         0.516811
5         0.515306
6         0.508929
7         0.482425
8         0.205367
9          0.20006
10         0.12964
11        0.123496
12        0.112628
13        0.110282
14        0.087622
15        0.071871
16        0.068313
17        0.062968
18        0.060664
19        0.055904
20        0.052733
21        0.048465
22        0.041519
23        0.040535
24         0.03164
25        0.019917
26        0.015628
27        0.011496
28        0.006411
29         0.00163
30       -0.002076
31       -0.014855
32       -0.015803
33               -
34               -
35               -
36               -
37               -
38               -
39               -
40               -
41               -
42               -
43               -
44               -
```

Looking at the importance of the features and conditions obtained from the rules determined by RuleKit, one can come to conclusions similar to those obtained for rules determined from the decision tree. The main difference is that in rules generated by RuleKit, the fare also plays an important role - the lower the fare, the person traveled in a lower class what determined their survival.

4. Creation of a binary dataset based on top conditions

[59]:
```
X_train_tranformed = explainer.fit_transform(X_train, selector=25)

X_train_tranformed.head(5)
```

[59]:
```
  Fare = <52.28, 143.59) Fare = <29.85, inf) Fare = <2.01, 7.13) Pclass = {1}  \
0                      1                   1                   0            1
1                      0                   0                   0            0
2                      0                   0                   0            0
3                      0                   0                   0            0
4                      0                   0                   0            0

  Age = (-inf, 0.92) Parch = <3.5, inf) Fare = <74.38, inf) Age = <2.5, 6.5)  \
0                  0                  0                   1                1
1                  0                  0                   0                0
2                  0                  0                   0                0
3                  0                  0                   0                0
4                  0                  0                   0                0

  Sex = {male} Pclass = {3} Fare = <10.48, inf) Fare = <9.41, 10.48)  \
0            1            0                   1                    0
1            1            1                   0                    0
2            0            1                   1                    0
3            1            0                   1                    0
4            1            0                   1                    0

  Fare = (-inf, 10.48) SibSp = <4.5, inf) Fare = (-inf, 7.13) Sex = {female}  \
0                    0                  0                   0              0
1                    1                  0                   0              0
2                    0                  0                   0              1
3                    0                  0                   0              0
4                    0                  0                   0              0

  Pclass = {2} Fare = <82.66, inf) Age = (-inf, 6.5)
0            0                   0                 1
1            0                   0                 0
2            0                   0                 1
3            1                   0                 0
4            1                   0                 0
```

[60]:
```
X_test_transformed = explainer.transform(X_test)

X_test_transformed.head(5)
```

[60]:
```
  Fare = <52.28, 143.59) Fare = <29.85, inf) Fare = <2.01, 7.13) Pclass = {1}  \
0                      0                   0                   0            0
1                      0                   0                   0            0
2                      0                   0                   0            0
3                      0                   1                   0            0
```

(continues on next page)

(continued from previous page)

```
4                    0                  0                  0            0

  Age = (-inf, 0.92) Parch = <3.5, inf) Fare = <74.38, inf) Age = <2.5, 6.5)  \
0                  0                  0                  0                0
1                  0                  0                  0                0
2                  0                  0                  0                0
3                  0                  0                  0                1
4                  0                  0                  0                0

  Sex = {male} Pclass = {3} Fare = <10.48, inf) Fare = <9.41, 10.48)  \
0            1            1                   1                    0
1            1            0                   1                    0
2            1            1                   0                    0
3            0            0                   1                    0
4            0            1                   1                    0

  Fare = (-inf, 10.48) SibSp = <4.5, inf) Fare = (-inf, 7.13) Sex = {female}  \
0                    0                  0                   0              0
1                    0                  0                   0              0
2                    1                  0                   0              0
3                    0                  0                   0              1
4                    0                  0                   0              1

  Pclass = {2} Fare = <82.66, inf) Age = (-inf, 6.5)
0            0                   0                  0
1            1                   0                  0
2            0                   0                  0
3            1                   0                  1
4            0                   0                  0
```

A binary dataset created in this way can be used to create a classifier using another machine learning algorithm. The advantage of this approach is that RuleKit has created rules on a set containing null values and containing both numeric and categorical variables. However, the prepared dataset consists only of numerical values 0 and 1 determining whether a given condition has been met and does not contain empty values. Therefore, you can easily use algorithms that deal only with numerical values and do not handle missing values, such as RandomForest as shown below.

[62]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=1)
cv = cross_val_score(rf, X_train_tranformed,
                     y_train, cv=5, scoring = "balanced_accuracy")
print(cv)
print(cv.mean())
```

```
[0.7570922  0.86241057 0.703082   0.73188406 0.8113155 ]
0.7731568645642581
```

[64]:
```
rf.fit(X_train_tranformed, y_train)
```

[64]:
```
RandomForestClassifier(random_state=1)
```

[65]:
```
balanced_accuracy_score(y_test, rf.predict(X_test_transformed))
```

[65]:
```
0.7861077638147702
```

VI. Summary

The presented analysis shows how the RuleXAI library may be used for data analysis and model explanation. Explanations, both global and local, are performed using the generated rule-based model representation.

[2]:
```
import pandas as pd
from scipy.io import arff
from rulekit import RuleKit
from rulekit.regression import RuleRegressor
from rulekit.params import Measures

from rulexai.explainer import RuleExplainer
```

1.3.2 CPU

Read data

[3]:
```
dataset_path = "./data/cpu.arff"
data = pd.DataFrame(arff.loadarff(dataset_path)[0])

# code to change encoding of the file
tmp_df = data.select_dtypes([object])
tmp_df = tmp_df.stack().str.decode("utf-8").unstack()
for col in tmp_df:
    data[col] = tmp_df[col].replace({"?": None})

x = data.drop(["class"], axis=1)
y = data["class"]
```

Train RuleKit model

[11]:
```
# RuleKit
RuleKit.init()

reg = RuleRegressor(
    induction_measure=Measures.C2,
    pruning_measure=Measures.C2,
    voting_measure=Measures.C2,
)
reg.fit(x, y)
```

[11]:
```
<rulekit.regression.RuleRegressor at 0x28bffccc670>
```

Rules

[12]:
```
for rule in reg.model.rules:
    print(rule, rule.stats)
```

```
IF vendor = {formation} THEN class = {34} [34,34] (p = 5.0, n = 0.0, P = 6.0, N = 203.0,␣
↪weight = 0.9166666666666667, pvalue = 0.0)
IF MMIN = <80, inf) AND MMAX = (-inf, 1750) THEN class = {18} [16.92,19.08] (p = 10.0, n␣
↪= 1.0, P = 11.0, N = 198.0, weight = 0.8629476584022039, pvalue = 7.355108555449812e-
↪21)
IF MMIN = <756, inf) AND MMAX = (-inf, 4250) AND CHMAX = <7, 22) AND CHMIN = (-inf, 3.
↪50) THEN class = {32} [30.64,33.36] (p = 4.0, n = 1.0, P = 7.0, N = 202.0, weight = 0.
↪6231258840169731, pvalue = 1.1803717269256882e-08)
IF MMIN = <756, inf) AND MMAX = (-inf, 4250) AND MYCT = (-inf, 232.50) AND CHMAX = <3.50,
↪ 22) AND CHMIN = (-inf, 3.50) THEN class = {29} [24.98,33.02] (p = 15.0, n = 3.0, P =␣
↪35.0, N = 174.0, weight = 0.5712917350848385, pvalue = 7.408462419973687e-25)
IF MMIN = (-inf, 1500) AND MMAX = <1500, 4250) AND MYCT = <94.50, inf) AND CHMAX = <2.50,
↪ 44) THEN class = {24} [21.77,26.23] (p = 18.0, n = 7.0, P = 23.0, N = 186.0, weight =␣
↪0.6108789153810191, pvalue = 1.183267277682215e-40)
IF MMAX = (-inf, 4750) THEN class = {24} [10.30,37.70] (p = 69.0, n = 2.0, P = 88.0, N =␣
↪121.0, weight = 0.8486424746828075, pvalue = 1.6425318084016525e-60)
IF MYCT = <87, inf) AND CHMAX = (-inf, 96) THEN class = {29} [1.17,56.83] (p = 107.0, n␣
↪= 11.0, P = 124.0, N = 85.0, weight = 0.7179513877721673, pvalue = 1.3893662585668293e-
↪64)
IF MMAX = <6150, 9240) AND MYCT = (-inf, 129) AND CACH = <2, 28) AND CHMAX = (-inf, 46)␣
↪THEN class = {46} [43.77,48.23] (p = 9.0, n = 2.0, P = 13.0, N = 196.0, weight = 0.
↪6821036106750392, pvalue = 1.023395667474569e-17)
IF MMIN = (-inf, 2150) AND MMAX = <5000, 9240) AND MYCT = (-inf, 146.50) AND CHMAX = <5.
↪50, inf) THEN class = {46} [14.85,77.15] (p = 25.0, n = 1.0, P = 143.0, N = 66.0,␣
↪weight = 0.5158687466379773, pvalue = 7.403283011266057e-14)
IF MMIN = <2310, 4500) AND MYCT = <31.50, 102.50) AND CACH = (-inf, 48) AND CHMAX = (-
↪inf, 40) THEN class = {80} [57.27,102.73] (p = 12.0, n = 2.0, P = 34.0, N = 175.0,␣
↪weight = 0.5610564225690277, pvalue = 1.34750514438087e-09)
IF MMIN = <640, 4500) AND MMAX = <7150, 24000) THEN class = {65} [36.20,93.80] (p = 60.0,
↪ n = 13.0, P = 68.0, N = 141.0, weight = 0.6927380687046022, pvalue = 2.
↪2589525624983582e-39)
IF MYCT = <27.50, 44) AND CHMIN = (-inf, 10) THEN class = {253} [192.76,313.24] (p = 7.0,
↪ n = 3.0, P = 12.0, N = 197.0, weight = 0.5396996615905246, pvalue = 0.
↪001963352522246969)
IF MMIN = <884, inf) AND MMAX = <9240, inf) AND CHMAX = <2.50, 88) AND CHMIN = (-inf,␣
↪14) THEN class = {117} [44.09,189.91] (p = 49.0, n = 11.0, P = 80.0, N = 129.0, weight␣
↪= 0.5667708333333334, pvalue = 4.475942404933969e-11)
IF MMIN = <3000, inf) AND MMAX = <24000, 48000) AND CHMIN = <14, inf) THEN class = {381}␣
↪[301.01,460.99] (p = 6.0, n = 1.0, P = 8.0, N = 201.0, weight = 0.7450248756218906,␣
↪pvalue = 0.047637666066025854)
IF MMIN = (-inf, 24000) AND MMAX = <28000, inf) AND MYCT = (-inf, 95) AND CACH = (-inf,␣
↪192) THEN class = {341} [129.60,552.40] (p = 19.0, n = 3.0, P = 34.0, N = 175.0,␣
↪weight = 0.6524789915966387, pvalue = 0.990671648706587)
```

RuleXAI

[13]:
```
explainer = RuleExplainer(model=reg, X=x, y=y, type="regression")
explainer.explain()
```

[13]:
```
<rulexai.explainer.RuleExplainer at 0x28ba8c77b50>
```

Feature importance

[14]:
```
explainer.feature_importances_
```

[14]:
```
  attributes  importances
3       MMAX     4.014332
2      CHMIN     3.028757
6     vendor     0.916667
1      CHMAX     0.460550
0       CACH     0.289558
4       MMIN     0.167137
5       MYCT    -1.233983
```

Condition importance

[15]:
```
explainer.condition_importances_
```

[15]:
```
                   conditions  importances
0         CHMIN = (-inf, 10.0)     2.127775
1         vendor = {formation}     0.916667
2        MMAX = (-inf, 4750.0)     0.848642
3        MMAX = (-inf, 1750.0)     0.827179
4           MYCT = <87.0, inf)     0.643064
5        MMAX = (-inf, 4250.0)     0.528220
6     MMAX = <7150.0, 24000.0)     0.481404
7         CHMIN = (-inf, 14.0)     0.402859
8        MMAX = <28000.0, inf)     0.381381
9    MMAX = <24000.0, 48000.0)     0.339882
10     MMAX = <6150.0, 9240.0)     0.307522
11         CHMIN = (-inf, 3.5)     0.260506
12         CHMIN = <14.0, inf)     0.237616
13     MMAX = <1500.0, 4250.0)     0.224479
14      MMIN = <640.0, 4500.0)     0.211334
15     MMAX = <5000.0, 9240.0)     0.198756
16       MMIN = (-inf, 1500.0)     0.198058
17       MMIN = (-inf, 2150.0)     0.185016
18          MYCT = <94.5, inf)     0.179675
19         CHMAX = <2.5, 88.0)     0.165561
20         CACH = (-inf, 48.0)     0.154017
21          CACH = <2.0, 28.0)     0.109025
22     MMIN = <2310.0, 4500.0)     0.090892
23        CHMAX = (-inf, 96.0)     0.074887
24        CHMAX = (-inf, 46.0)     0.066936
25         CHMAX = <7.0, 22.0)     0.062233
```

(continues on next page)

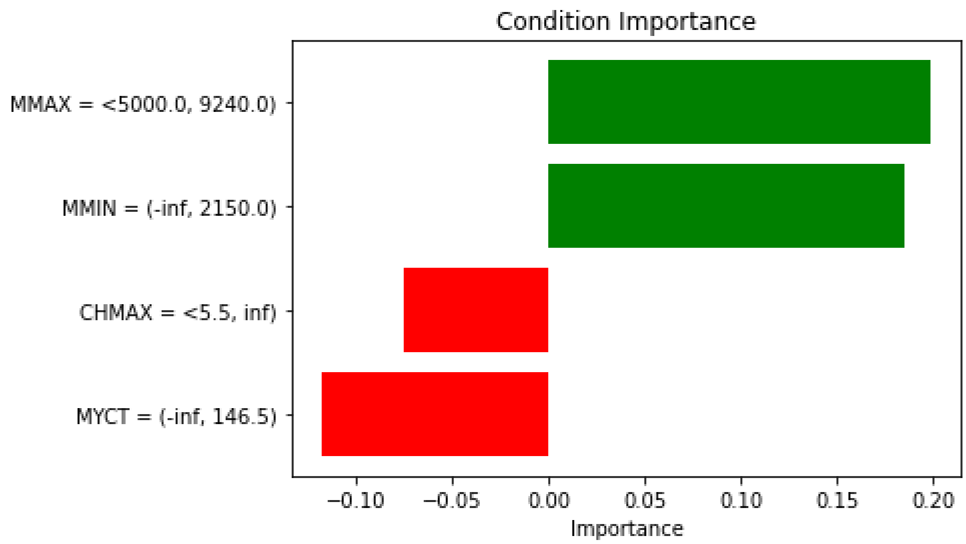
(continued from previous page)

```
26       CHMAX = (-inf, 40.0)     0.059474
27        CHMAX = <3.5, 22.0)     0.056674
28     MMIN = (-inf, 24000.0)     0.054221
29        CHMAX = <2.5, 44.0)     0.049421
30         MMIN = <80.0, inf)     0.035768
31       CACH = (-inf, 192.0)     0.026516
32       MYCT = <31.5, 102.5)     0.026372
33        MMIN = <756.0, inf)     0.003292
34       MYCT = (-inf, 232.5)    -0.033761
35        MMIN = <884.0, inf)    -0.069930
36         CHMAX = <5.5, inf)    -0.074637
37       MYCT = (-inf, 146.5)    -0.117779
38       MMAX = <9240.0, inf)    -0.123134
39       MYCT = (-inf, 129.0)    -0.151520
40        MYCT = (-inf, 95.0)    -0.191957
41       MMIN = <3000.0, inf)    -0.541514
42        MYCT = <27.5, 44.0)    -1.588076
```

Local explainability

[16]:
```
explainer.local_explainability(x.iloc[0, :], pd.DataFrame(y).iloc[0, :], plot = True)
```

```
Example:
vendor    adviser
MYCT        125.0
MMIN        256.0
MMAX       6000.0
CACH        256.0
CHMIN        16.0
CHMAX       128.0
class       199.0
Name: 0, dtype: object

Rules that covers this example:
IF MMIN = (-inf, 2150.0) AND MMAX = <5000.0, 9240.0) AND MYCT = (-inf, 146.5) AND CHMAX␣
↪= <5.5, inf) THEN class = {46.0}

Importances of the conditions from rules covering the example
                conditions  importances
0  MMAX = <5000.0, 9240.0)     0.198756
1    MMIN = (-inf, 2150.0)     0.185016
2       CHMAX = <5.5, inf)    -0.074637
3     MYCT = (-inf, 146.5)    -0.117779
```



[16]:

```
              conditions  importances
0  MMAX = <5000.0, 9240.0)     0.198756
1    MMIN = (-inf, 2150.0)     0.185016
2       CHMAX = <5.5, inf)    -0.074637
3     MYCT = (-inf, 146.5)    -0.117779
```

[2]:

```python
import pandas as pd
from scipy.io import arff
from rulekit import RuleKit
from rulekit.survival import SurvivalRules
from rulekit.params import Measures

from rulexai.explainer import RuleExplainer
```

1.3.3 GBSG2

Read data

[3]:

```python
dataset_path = "./data/GBSG2.arff"
data = pd.DataFrame(arff.loadarff(dataset_path)[0])

# code to change encoding of the file
tmp_df = data.select_dtypes([object])
tmp_df = tmp_df.stack().str.decode("utf-8").unstack()
for col in tmp_df:
    data[col] = tmp_df[col].replace({"?": None})

x = data.drop(["survival_status"], axis=1)
y = data["survival_status"]
```

Train RuleKit model

[4]:
```
# RuleKit
RuleKit.init()

srv = SurvivalRules(survival_time_attr="survival_time")
srv.fit(values=x, labels=y)
```

[4]:
```
<rulekit.survival.SurvivalRules at 0x176db91a880>
```

Rules

[5]:
```
for rule in srv.model.rules:
    print(rule, rule.stats)
```

```
IF pnodes = (-inf, 3.50) THEN survival_status = {NaN} (p = 304.0, n = 0.0, P = 564.0, N␣
↪= 0.0, weight = 0.9999999999999998, pvalue = 2.220446049250313e-16)
IF pnodes = (-inf, 17.50) AND progrec = (-inf, 9.50) AND age = <41.50, 52.50) AND estrec␣
↪= <0.50, 29) THEN survival_status = {NaN} (p = 21.0, n = 0.0, P = 564.0, N = 0.0,␣
↪weight = 0.9999999999909083, pvalue = 9.09172737095787e-12)
IF pnodes = <4.50, 19) AND progrec = (-inf, 11.50) AND age = <41.50, 64.50) AND estrec =
↪<0.50, 41) THEN survival_status = {NaN} (p = 33.0, n = 0.0, P = 564.0, N = 0.0, weight␣
↪= 1.0, pvalue = 0.0)
IF pnodes = <4.50, inf) AND progrec = (-inf, 25.50) THEN survival_status = {NaN} (p =␣
↪113.0, n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
IF pnodes = <4.50, inf) AND progrec = (-inf, 99) THEN survival_status = {NaN} (p = 156.0,
↪ n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
IF pnodes = <5.50, inf) AND progrec = (-inf, 135) THEN survival_status = {NaN} (p = 144.
↪0, n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
IF pnodes = <4.50, inf) AND progrec = (-inf, 233) THEN survival_status = {NaN} (p = 185.
↪0, n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
IF pnodes = (-inf, 4.50) AND progrec = <9, inf) AND age = <39.50, inf) THEN survival_
↪status = {NaN} (p = 245.0, n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
IF progrec = <107, inf) THEN survival_status = {NaN} (p = 168.0, n = 0.0, P = 564.0, N =␣
↪0.0, weight = 0.9999999989621143, pvalue = 1.0378856662995872e-09)
IF pnodes = <3.50, inf) AND progrec = (-inf, 105.50) THEN survival_status = {NaN} (p =␣
↪195.0, n = 0.0, P = 564.0, N = 0.0, weight = 1.0, pvalue = 0.0)
```

RuleXAI

[6]:
```
explainer = RuleExplainer(model=srv, X=x, y=y, type="survival")
explainer.explain()
```

[6]:
```
<rulexai.explainer.RuleExplainer at 0x176db937700>
```

Feature importance

[7]:
```
explainer.feature_importances_
```

[7]:
```
  attributes  importances
2     pnodes   460.222804
3    progrec   251.499862
0        age    20.523849
1     estrec    13.347720
```

Condition importance

[8]:
```
explainer.condition_importances_
```

[8]:
```
                   conditions  importances
0       pnodes = <4.5, inf)   207.268572
1      pnodes = (-inf, 3.5)    67.394775
2       pnodes = <5.5, inf)    64.254026
3       pnodes = <3.5, inf)    64.104973
4    progrec = (-inf, 25.5)    48.923100
5    progrec = <107.0, inf)    37.252374
6   progrec = (-inf, 105.5)    33.962572
7    progrec = (-inf, 99.0)    33.423755
8      pnodes = (-inf, 4.5)    32.835122
9   progrec = (-inf, 135.0)    25.353218
10   progrec = (-inf, 11.5)    23.663185
11    progrec = (-inf, 9.5)    23.506762
12     pnodes = <4.5, 19.0)    18.150272
13     progrec = <9.0, inf)    13.146344
14  progrec = (-inf, 233.0)    12.268552
15     estrec = <0.5, 29.0)    10.450381
16       age = <41.5, 64.5)     9.275232
17       age = <41.5, 52.5)     8.077389
18    pnodes = (-inf, 17.5)     6.215064
19        age = <39.5, inf)     3.171229
20     estrec = <0.5, 41.0)     2.897339
```

Local explainability

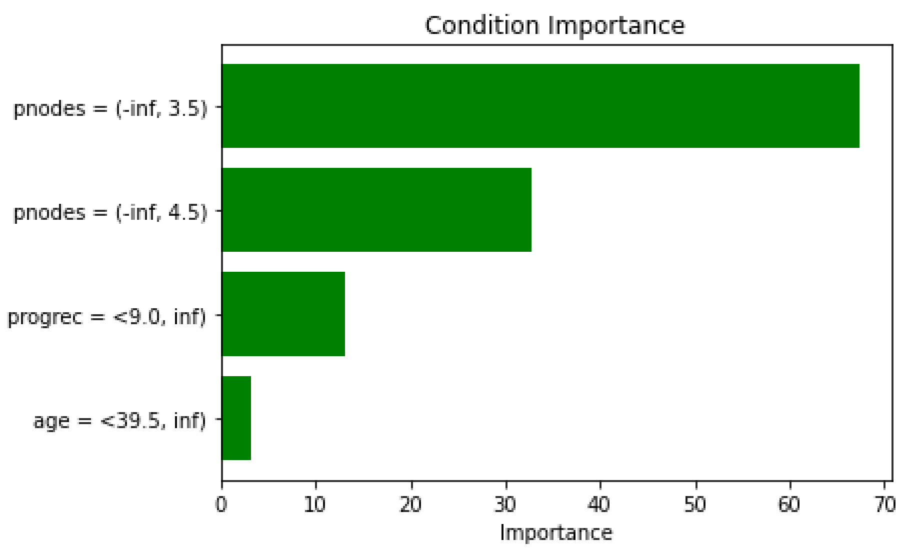
[9]:
```
explainer.local_explainability(x.iloc[0, :], pd.DataFrame(y).iloc[0, :], plot = True)
```

```
Example:
horTh                  no
age                  70.0
menostat             Post
tsize                21.0
tgrade                 II
```

(continues on next page)

(continued from previous page)

```
pnodes                 3.0
progrec               48.0
estrec                66.0
survival_time       1814.0
survival_status        1.0
Name: 0, dtype: object

Rules that covers this example:
IF pnodes = (-inf, 3.5) THEN survival_status = {NaN}
IF pnodes = (-inf, 4.5) AND progrec = <9.0, inf) AND age = <39.5, inf) THEN survival_
↪status = {NaN}

Importances of the conditions from rules covering the example
             conditions  importances
0  pnodes = (-inf, 3.5)    67.394775
1  pnodes = (-inf, 4.5)    32.835122
2  progrec = <9.0, inf)    13.146344
3     age = <39.5, inf)     3.171229
```



[9]:

```
             conditions  importances
0  pnodes = (-inf, 3.5)    67.394775
1  pnodes = (-inf, 4.5)    32.835122
2  progrec = <9.0, inf)    13.146344
3     age = <39.5, inf)     3.171229
```

1.3.4 Black-box model aproximation

The purpose of this notebook is to demonstrate the possibility of using RuleXAI to explain black box models. The data set titanic from OpenML (https://www.openml.org/d/40945) was used in the analysis. It is a popular data set often used in various types of examples, therefore it was decided to use it in this analysis.

Data load

[48]:
```
import pandas as pd

data = pd.read_csv('./data/titanic_openml.csv')
data
```

[48]:
```
      pclass  survived                                             name  \
0          1         1                    Allen, Miss. Elisabeth Walton
1          1         1                   Allison, Master. Hudson Trevor
2          1         0                     Allison, Miss. Helen Loraine
3          1         0             Allison, Mr. Hudson Joshua Creighton
4          1         0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
...      ...       ...                                              ...
1304       3         0                             Zabour, Miss. Hileni
1305       3         0                            Zabour, Miss. Thamine
1306       3         0                        Zakarian, Mr. Mapriededer
1307       3         0                              Zakarian, Mr. Ortin
1308       3         0                               Zimmerman, Mr. Leo

         sex      age  sibsp  parch  ticket      fare    cabin embarked boat  \
0     female  29.0000      0      0   24160  211.3375       B5        S    2
1       male   0.9167      1      2  113781  151.5500  C22 C26        S   11
2     female   2.0000      1      2  113781  151.5500  C22 C26        S  NaN
3       male  30.0000      1      2  113781  151.5500  C22 C26        S  NaN
4     female  25.0000      1      2  113781  151.5500  C22 C26        S  NaN
...      ...      ...    ...    ...     ...       ...      ...      ...  ...
1304  female  14.5000      1      0    2665   14.4542      NaN        C  NaN
1305  female      NaN      1      0    2665   14.4542      NaN        C  NaN
1306    male  26.5000      0      0    2656    7.2250      NaN        C  NaN
1307    male  27.0000      0      0    2670    7.2250      NaN        C  NaN
1308    male  29.0000      0      0  315082    7.8750      NaN        S  NaN

       body                        home.dest
0       NaN                     St Louis, MO
1       NaN  Montreal, PQ / Chesterville, ON
2       NaN  Montreal, PQ / Chesterville, ON
3     135.0  Montreal, PQ / Chesterville, ON
4       NaN  Montreal, PQ / Chesterville, ON
...     ...                              ...
1304  328.0                              NaN
1305    NaN                              NaN
1306  304.0                              NaN
1307    NaN                              NaN
1308    NaN                              NaN

[1309 rows x 14 columns]
```

Dataset overwiev

[49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   int64
 1   survived   1309 non-null   int64
 2   name       1309 non-null   object
 3   sex        1309 non-null   object
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   int64
 6   parch      1309 non-null   int64
 7   ticket     1309 non-null   object
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object
 10  embarked   1307 non-null   object
 11  boat       486 non-null    object
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object
dtypes: float64(3), int64(4), object(7)
memory usage: 143.3+ KB
```

[50]:
```
data[['survived', 'pclass']
     ] = data[['survived', 'pclass']].astype(str)
```

[51]:
```
numeric_data = data[['age', 'sibsp', 'parch', 'fare', 'body']]
caterogical_data = data[['name', 'sex', 'ticket',
                         'cabin', 'embarked', 'boat', 'home.dest']]
```

[52]:
```
numeric_data.describe()
```

[52]:
```
               age        sibsp        parch         fare        body
count  1046.000000  1309.000000  1309.000000  1308.000000  121.000000
mean     29.881135     0.498854     0.385027    33.295479  160.809917
std      14.413500     1.041658     0.865560    51.758668   97.696922
min       0.166700     0.000000     0.000000     0.000000    1.000000
25%      21.000000     0.000000     0.000000     7.895800   72.000000
50%      28.000000     0.000000     0.000000    14.454200  155.000000
75%      39.000000     1.000000     0.000000    31.275000  256.000000
max      80.000000     8.000000     9.000000   512.329200  328.000000
```

[53]:
```
caterogical_data.describe()
```

[53]:
```
                     name   sex    ticket        cabin embarked boat  \
count                1309  1309      1309          295     1307  486
unique               1307     2       929          186        3   27
top     Kelly, Mr. James  male  CA. 2343  C23 C25 C27        S   13
freq                    2   843        11            6      914   39

```

(continues on next page)

(continued from previous page)

```
         home.dest
count          745
unique         369
top    New York, NY
freq            64
```

Data preprocessing

In the first stage of data preprocessing it was decided to only remove the columns for Passenger Name, ticket type, cabin, embarked, boat, home.dest, body. Removing the Passenger Name columns is self-explanatory - in no way does Passenger Name have any bearing on whether a person survived. It would only be possible to derive passenger status from passenger name, as there are markings such as 'Mr.', 'Mrs.', 'Miss.', 'Master.'. In case of tickets, the designations for most tickets vary - 681 unique values out of 891 occurrences. One could extract some information from the tickets from their designations (e.g., whether they begin with a number or a letter). However, you would have to consult historical data to find out what the ticket designations mean. In the case of cabin designations, as many as 697 values are missing - for this reason it was decided to remove the entire column, as it carries too little information. On the basis of a similar analysis, the remaining mentioned columns were removed

Of course, the preliminary data analysis and preprocessing stage itself could have been even more extensive - exploring the relationships between features, examining the impact of individual features, plotting graphs to better understand the data. However, the main purpose of this notebook is not to analyze a given set of data in detail, but only to show the possibilities of using the RuleXAI library. For this reason, some simplifications in the analysis have been decided.

[54]:
```
data.drop(["name", "ticket", "cabin", "embarked", "boat", "home.dest", "body"], axis=1,␣
↪inplace=True)
data.reset_index(inplace=True, drop=True)

data.head(5)
```

[54]:

	pclass	survived	sex	age	sibsp	parch	fare
0	1	1	female	29.0000	0	0	211.3375
1	1	1	male	0.9167	1	2	151.5500
2	1	0	female	2.0000	1	2	151.5500
3	1	0	male	30.0000	1	2	151.5500
4	1	0	female	25.0000	1	2	151.5500

Building black-box model - neural network

In order to demonstrate the possibility of using the RuleXAI library to explain black-box models, it was decided to use the Titanic set to build a neural network to classify whether a given person survived or not. Then, with the help of the RuleXAI library, an analysis will be performed to explain on what basis the neural network model makes decisions.

Since neural networks do not handle missing data and operate only on numerical data, it was necessary to fill in the missing data, perform dummification and scaling.

[55]:
```
from sklearn.preprocessing import MinMaxScaler

data.age = data.age.fillna(data.age.median())
data.fare = data.age.fillna(data.fare.median())

scaler = MinMaxScaler()
```

(continues on next page)

(continued from previous page)

```
data_dummies = pd.get_dummies(data.drop(["survived"], axis=1))
data_dummies = data_dummies.drop(["sex_male"], axis=1)
data_scaled = pd.DataFrame(scaler.fit_transform(data_dummies),index=data_dummies.index,
↪columns=data_dummies.columns)

X = data_scaled
y = data.survived.astype(int)

X.head(5)
```

[55]:

```
        age  sibsp     parch      fare  pclass_1  pclass_2  pclass_3  \
0  0.361169  0.000  0.000000  0.361169       1.0       0.0       0.0
1  0.009395  0.125  0.222222  0.009395       1.0       0.0       0.0
2  0.022964  0.125  0.222222  0.022964       1.0       0.0       0.0
3  0.373695  0.125  0.222222  0.373695       1.0       0.0       0.0
4  0.311064  0.125  0.222222  0.311064       1.0       0.0       0.0

   sex_female
0         1.0
1         0.0
2         1.0
3         0.0
4         1.0
```

[56]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
↪ )
```

Neural Network learning

[66]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Input(shape=(X_train.shape[1],)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation="relu"))
model.add(Dense(32, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(X_train.to_numpy(), y_train.to_numpy(), epochs=100, batch_size = 24,␣
↪validation_split=0.2, callbacks=[early_stopping_callback])
```

```
Epoch 1/100
31/31 [==============================] - 0s 16ms/step - loss: 0.6153 - accuracy: 0.7254 -
↪ val_loss: 0.5677 - val_accuracy: 0.7446
```

(continues on next page)

(continued from previous page)

```
Epoch 2/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4747 - accuracy: 0.7964 -␣
↪val_loss: 0.5232 - val_accuracy: 0.7500
Epoch 3/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4416 - accuracy: 0.8033 -␣
↪val_loss: 0.5081 - val_accuracy: 0.7500
Epoch 4/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4318 - accuracy: 0.8046 -␣
↪val_loss: 0.5271 - val_accuracy: 0.7500
Epoch 5/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4359 - accuracy: 0.8074 -␣
↪val_loss: 0.4926 - val_accuracy: 0.7446
Epoch 6/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4364 - accuracy: 0.8033 -␣
↪val_loss: 0.4932 - val_accuracy: 0.7446
Epoch 7/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4268 - accuracy: 0.8156 -␣
↪val_loss: 0.4931 - val_accuracy: 0.7554
Epoch 8/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4242 - accuracy: 0.8183 -␣
↪val_loss: 0.4920 - val_accuracy: 0.7554
Epoch 9/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4238 - accuracy: 0.8060 -␣
↪val_loss: 0.4882 - val_accuracy: 0.7609
Epoch 10/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4232 - accuracy: 0.8046 -␣
↪val_loss: 0.4997 - val_accuracy: 0.7609
Epoch 11/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4201 - accuracy: 0.8046 -␣
↪val_loss: 0.4876 - val_accuracy: 0.7717
Epoch 12/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4225 - accuracy: 0.8005 -␣
↪val_loss: 0.4904 - val_accuracy: 0.7609
Epoch 13/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4166 - accuracy: 0.8169 -␣
↪val_loss: 0.5062 - val_accuracy: 0.7609
Epoch 14/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4222 - accuracy: 0.8183 -␣
↪val_loss: 0.4938 - val_accuracy: 0.7554
Epoch 15/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4158 - accuracy: 0.8279 -␣
↪val_loss: 0.5170 - val_accuracy: 0.7609
Epoch 16/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4236 - accuracy: 0.8156 -␣
↪val_loss: 0.4919 - val_accuracy: 0.7609
Epoch 17/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4185 - accuracy: 0.8128 -␣
↪val_loss: 0.5000 - val_accuracy: 0.7500
Epoch 18/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4149 - accuracy: 0.8156 -␣
↪val_loss: 0.4982 - val_accuracy: 0.7609
Epoch 19/100
```

(continues on next page)

(continued from previous page)

```
31/31 [==============================] - 0s 7ms/step - loss: 0.4131 - accuracy: 0.8210 -␣
↪val_loss: 0.4961 - val_accuracy: 0.7717
Epoch 20/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4136 - accuracy: 0.8156 -␣
↪val_loss: 0.4820 - val_accuracy: 0.7717
Epoch 21/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4162 - accuracy: 0.8169 -␣
↪val_loss: 0.4970 - val_accuracy: 0.7717
Epoch 22/100
31/31 [==============================] - 0s 8ms/step - loss: 0.4184 - accuracy: 0.8169 -␣
↪val_loss: 0.4944 - val_accuracy: 0.7772
Epoch 23/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4112 - accuracy: 0.8224 -␣
↪val_loss: 0.4861 - val_accuracy: 0.7717
Epoch 24/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4156 - accuracy: 0.8183 -␣
↪val_loss: 0.4895 - val_accuracy: 0.7772
Epoch 25/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4107 - accuracy: 0.8087 -␣
↪val_loss: 0.4933 - val_accuracy: 0.7717
Epoch 26/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4077 - accuracy: 0.8183 -␣
↪val_loss: 0.4903 - val_accuracy: 0.7717
Epoch 27/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4141 - accuracy: 0.8128 -␣
↪val_loss: 0.5159 - val_accuracy: 0.7772
Epoch 28/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4117 - accuracy: 0.8197 -␣
↪val_loss: 0.4859 - val_accuracy: 0.7717
Epoch 29/100
31/31 [==============================] - 0s 8ms/step - loss: 0.4089 - accuracy: 0.8238 -␣
↪val_loss: 0.4802 - val_accuracy: 0.7554
Epoch 30/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4084 - accuracy: 0.8197 -␣
↪val_loss: 0.5109 - val_accuracy: 0.7717
Epoch 31/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4086 - accuracy: 0.8251 -␣
↪val_loss: 0.4889 - val_accuracy: 0.7717
Epoch 32/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4055 - accuracy: 0.8183 -␣
↪val_loss: 0.4891 - val_accuracy: 0.7609
Epoch 33/100
31/31 [==============================] - 0s 6ms/step - loss: 0.4055 - accuracy: 0.8156 -␣
↪val_loss: 0.5027 - val_accuracy: 0.7826
Epoch 34/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4099 - accuracy: 0.8224 -␣
↪val_loss: 0.5104 - val_accuracy: 0.7609
Epoch 35/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4055 - accuracy: 0.8156 -␣
↪val_loss: 0.4831 - val_accuracy: 0.7609
Epoch 36/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4027 - accuracy: 0.8197 -␣
↪val_loss: 0.4873 - val_accuracy: 0.7663
```

(continues on next page)

(continued from previous page)

```
Epoch 37/100
31/31 [==============================] - 0s 5ms/step - loss: 0.4069 - accuracy: 0.8265 -␣
↪val_loss: 0.4992 - val_accuracy: 0.7663
Epoch 38/100
31/31 [==============================] - 0s 7ms/step - loss: 0.4013 - accuracy: 0.8251 -␣
↪val_loss: 0.4923 - val_accuracy: 0.7717
Epoch 39/100
31/31 [==============================] - 0s 8ms/step - loss: 0.4010 - accuracy: 0.8265 -␣
↪val_loss: 0.4875 - val_accuracy: 0.7500
```

[14]:
```
model.save("./models/nn", save_format = 'h5')
```

Model evaluation on training and test set

[67]:
```
from sklearn.metrics import balanced_accuracy_score, accuracy_score
import numpy as np

train_acc = np.round(accuracy_score(y_train, model.predict(X_train)>0.5),3)
train_bacc = np.round(balanced_accuracy_score(y_train, model.predict(X_train)>0.5),3)

print(f"NN model train accuracy: {train_acc}")
print(f"NN model train bacc: {train_bacc}")

test_acc = np.round(accuracy_score(y_test, model.predict(X_test)>0.5),3)
test_bacc = np.round(balanced_accuracy_score(y_test, model.predict(X_test)>0.5),3)

print(f"NN model test accuracy: {test_acc}")
print(f"NN model test bacc: {test_bacc}")
```

```
NN model train accuracy: 0.816
NN model train bacc: 0.793
NN model test accuracy: 0.809
NN model test bacc: 0.794
```

Since the purpose of the analysis is not to create the best possible black-box model, but only to show the possibility of its explanation, it was concluded that the model obtaining a balanced accuracy of 0.793 on the training set and 0.794 on the test set is sufficient. Of course, testing other network architectures would yield better results, but that is not the purpose of this notebook.

RuleXAI

The RuleXAI library enables the explanation of black-box models by approximating the black-box model with a rule model. This is possible by replacing the decision variable in the dataset with decisions made by the network and teaching the rule model on that dataset. The rule-based model will therefore learn to map the data set to the decisions made by the black-box model. It is also worth noting that the rule-based model can then be trained on the original set (containing nominal and missing attributes). Such a procedure will facilitate the analysis. Instead of the conditions Sex_female = {0}, the condition set will have the condition Sex = {male}

[68]:
```
import numpy as np
```

(continues on next page)

(continued from previous page)

```
y_train_nn_decisions = np.array(list(map(int, model.predict(X_train)>0.5)))
y_test_nn_decisions = np.array(list(map(int, model.predict(X_test)>0.5)))

y_train_nn_df = pd.DataFrame(y_train_nn_decisions, columns=["label"]).astype(str)

X_org = data.drop(["survived"], axis=1)
y_org = data.survived

X_train_org = X_org.loc[X_train.index,:]
X_test_org = X_org.loc[X_test.index,:]

X_train_org.reset_index(inplace=True, drop=True)
X_train_org.head(5)
```

[68]:

	pclass	sex	age	sibsp	parch	fare
0	3	male	28.0	0	0	28.0
1	3	male	26.0	0	0	26.0
2	2	female	19.0	0	0	19.0
3	3	female	28.0	8	2	28.0
4	3	female	28.0	0	0	28.0

[101]:

```
from rulexai.explainer import Explainer

explainer =  Explainer(X = X_train,model_predictions = y_train_nn_df,type =
↪"classification")
```

[102]:

```
explainer.explain(X_org=X_train_org)
```

[102]:

```
<rulexai.explainer.Explainer at 0x18b72d3a640>
```

The approach to explaining black-box models with rule models is often already considered as the explainability of such models. When analyzing the resulting rules, certain conclusions can be drawn. The use of the RuleXAI library allows to go a step further - obtaining information about the importance of features and specific ranges of these features. This will enable a more in-depth analysis of the dataset and the black-box model.

Rules describing the black-box model

[91]:

```
for rule in explainer.get_rules():
    print(rule)
```

```
IF sex = {male} AND age = <8.5, 47.5) THEN label = {0}
IF sex = {male} AND age = <8.5, inf) THEN label = {0}
IF sex = {male} AND age = <4.5, 47.5) THEN label = {0}
IF pclass = {3} AND sibsp = <1.5, inf) AND age = <0.96, inf) THEN label = {0}
IF parch = <4.5, inf) THEN label = {0}
IF pclass = {3} AND sibsp = <0.5, inf) AND age = <28.25, inf) AND parch = <0.5, inf)␣
↪THEN label = {0}
IF pclass = {3} AND age = <27.5, 44.5) AND parch = <0.5, inf) THEN label = {0}
IF sex = {female} AND sibsp = (-inf, 1.5) AND parch = (-inf, 2.5) THEN label = {1}
IF sex = {female} AND sibsp = (-inf, 2.5) AND parch = (-inf, 3.5) THEN label = {1}
IF age = (-inf, 0.96) THEN label = {1}
IF sibsp = (-inf, 2.5) AND age = (-inf, 4.5) THEN label = {1}
IF pclass = {1} AND sibsp = <1.5, inf) THEN label = {1}
```

(continues on next page)

(continued from previous page)

```
IF pclass = {1} AND sibsp = <0.5, inf) AND age = <46.5, inf) AND parch = (-inf, 0.5)␣
↪THEN label = {1}
IF sibsp = (-inf, 3.5) AND parch = (-inf, 4.5) AND age = (-inf, 60.25) THEN label = {1}
```

Quality of the black-box model approximation

[103]:
```
rc = explainer.model.model

train_acc = np.round(accuracy_score(y_train_nn_decisions, rc.predict(X_train_org).
↪astype(int)),3)
train_bacc = np.round(balanced_accuracy_score(y_train_nn_decisions, rc.predict(X_train_
↪org).astype(int)),3)

print(f"Rule model train accuracy: {train_acc}")
print(f"Rule model train bacc: {train_bacc}")

test_acc = np.round(accuracy_score(y_test_nn_decisions, rc.predict(X_test_org).
↪astype(int)),3)
test_bacc = np.round(balanced_accuracy_score(y_test_nn_decisions, rc.predict(X_test_org).
↪astype(int)),3)

print(f"Rule model test accuracy: {test_acc}")
print(f"Rule model test bacc: {test_bacc}")
```

```
Rule model train accuracy: 0.971
Rule model train bacc: 0.968
Rule model test accuracy: 0.964
Rule model test bacc: 0.966
```

Rule condition importance

[94]:
```
explainer.condition_importances_
```

[94]:

	0 | conditions_names	0 | importances	1 | conditions_names	1 | importances
0	sex = {male}	2.506576	sex = {female}	1.095350
1	pclass = {3}	0.532272	age = (-inf, 0.96)	0.516234
2	parch = <4.5, inf)	0.506579	age = (-inf, 4.5)	0.449048
3	sibsp = <1.5, inf)	0.156118	pclass = {1}	0.428375
4	age = <27.5, 44.5)	0.131756	sibsp = (-inf, 2.5)	0.126061
5	age = <8.5, 47.5)	0.08452	sibsp = <0.5, inf)	0.111988
6	age = <4.5, 47.5)	0.07646	sibsp = <1.5, inf)	0.092592
7	age = <8.5, inf)	0.057054	age = <46.5, inf)	0.089966
8	age = <28.25, inf)	0.041569	sibsp = (-inf, 1.5)	0.031061
9	age = <0.96, inf)	0.026228	parch = (-inf, 2.5)	0.012163
10	sibsp = <0.5, inf)	-0.014897	parch = (-inf, 0.5)	0.011806
11	parch = <0.5, inf)	-0.052528	parch = (-inf, 3.5)	0.011632
12	-	-	sibsp = (-inf, 3.5)	0.010848
13	-	-	parch = (-inf, 4.5)	0.003102
14	-	-	age = (-inf, 60.25)	0.002035

Looking at the ranking of conditions obtained with the help of the RuleXAI library, it can be noticed that the greatest influence on the decision made by the black-box model as to whether a given person survived or not was gender. The most important condition indicating that the person survived is Sex = {female}, and the most important condition indicating that the person did not survive is Sex = {male}. It is intuitive and logical - the first women were rescued. Then it can be seen that the age of the person had a big impact on whether the person survived - children aged 0 to 4.5 had a greater chance of survival. Looking at the ranking on the impact of conditions on the fact that a given person did not survive, it can be seen that apart from the fact that the person was a man, it was also influenced by the fact that they traveled in 3rd class - it is also consistent with historical knowledge and logic - the first rescued there were more affluent people.

Feature importance

[105]:
```
explainer.feature_importances_
```

[105]:
```
  0 | attributes  0 | importances 1 | attributes  1 | importances
0            sex          2.506576            sex          1.095350
1         pclass          0.532272            age          1.057283
2          parch          0.454051         pclass          0.428375
3            age          0.417587          sibsp          0.372549
4          sibsp          0.141221          parch          0.038703
```

Looking at the global ranking of features importance, it can be seen that the most important features that influenced whether a person survived or not were gender, age, and the class they traveled. It is intuitive.

Local explainability

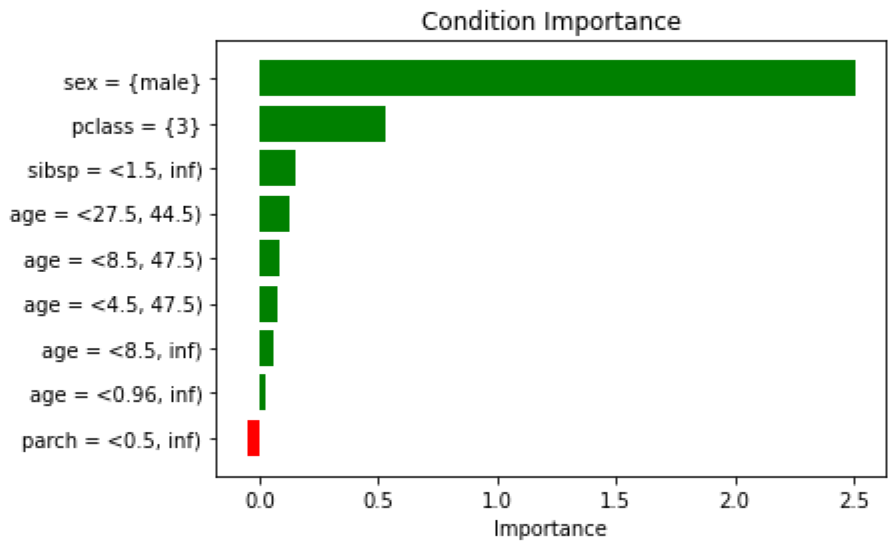
[75]:
```
y_test_nn_df = pd.DataFrame(y_test_nn_decisions, columns=["label"])
X_test_org.reset_index(inplace=True, drop=True)

local_explainability = explainer.local_explainability(X_test_org.iloc[10, :], y_test_nn_
↪df.iloc[10, :], plot = True)
```

```
Example:
pclass       3
sex       male
age       28.0
sibsp        8
parch        2
fare      28.0
label        0
Name: 10, dtype: object

Rules that covers this example:
IF sex = {male} AND age = <8.5, 47.5) THEN label = {0}
IF sex = {male} AND age = <8.5, inf) THEN label = {0}
IF sex = {male} AND age = <4.5, 47.5) THEN label = {0}
IF pclass = {3} AND sibsp = <1.5, inf) AND age = <0.96, inf) THEN label = {0}
IF pclass = {3} AND age = <27.5, 44.5) AND parch = <0.5, inf) THEN label = {0}

Importances of the conditions from rules covering the example
  0 | conditions_names 0 | importances
0          sex = {male}        2.506576
1          pclass = {3}        0.532272
2   sibsp = <1.5, inf)         0.156118
```

(continues on next page)

(continued from previous page)

```
3   age = <27.5, 44.5)        0.131756
4    age = <8.5, 47.5)         0.08452
5    age = <4.5, 47.5)         0.07646
6     age = <8.5, inf)        0.057054
7    age = <0.96, inf)        0.026228
8   parch = <0.5, inf)       -0.052528
```



Looking at the local explainability for an example from a test set, returned by the RuleXAI library, it can be seen what rules explaining the black-box model cover the given example. The chart of the importance of the conditions also shows that the condition Sex = {male} had the greatest influence on the model making such a decision. Subsequently, the fact that a given person did not survive was due to the fact that they traveled 3rd class and had more than 1 relative on board

SHAP

In order to compare the results and conclusions of the explainability of the black-box model with the help of the RuleXAI library, it was decided to explain the model also with the help of the SHAP library [https://shap.readthedocs.io/en/latest/index.html]. The SHAP library is one of the currently most popular and widely used libraries for black-box model explainability.

[106]:

```
import shap
shap.initjs()
```

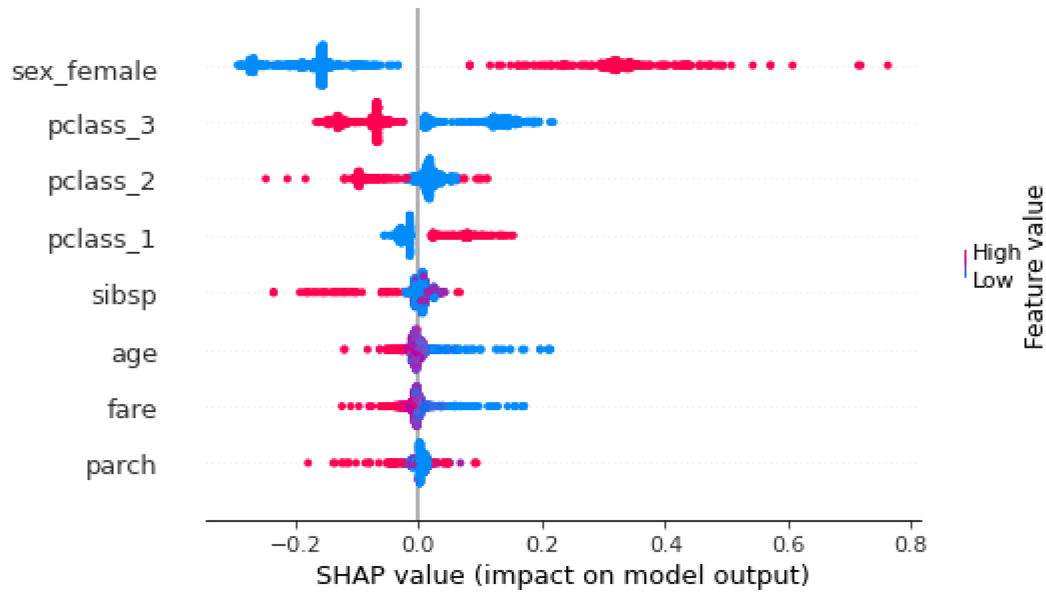
```
<IPython.core.display.HTML object>
```

[107]:

```
explainer = shap.DeepExplainer(model,X_train)
shap_values = explainer.shap_values(X_train.values)
```

Global ranking

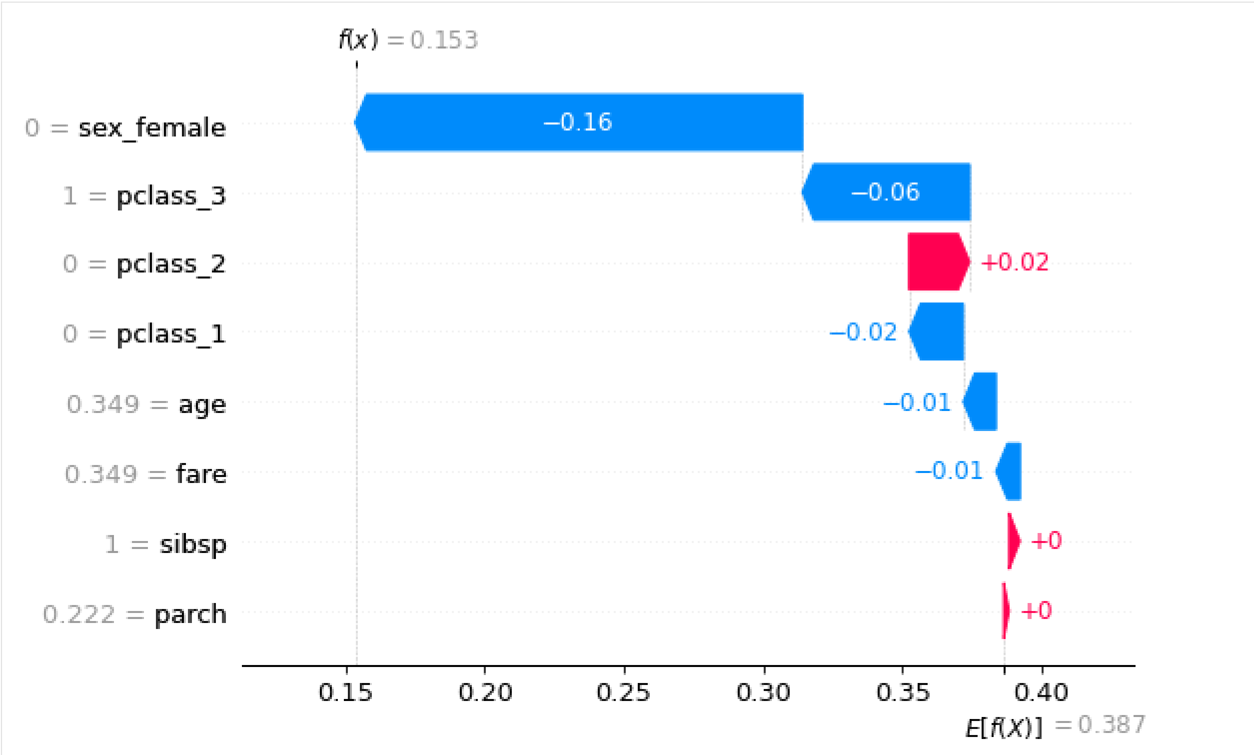
[108]:
```
shap.summary_plot(shap_values[0], X_train)
```



Comparing the ranking obtained using the SHAP library with the rankings obtained using the RuleXAI library, similar conclusions can be reached. The biggest influence on whether a person survived was whether person was female or not. The next most important attributes concern which class the person traveled in. These conclusions are in line with those drawn on the basis of the ranking obtained with the RuleXAI library.

Local explainability

[109]:
```
shap_values = explainer.shap_values(X_test.values)
shap.plots.waterfall(shap.Explanation(values=shap_values[0][0],
                base_values=np.array(explainer.expected_value)[0],
                data=X_test.iloc[10],
                feature_names=X_test.columns.tolist()))
```



Comparing the local explainability for the same example obtained using the RuleXAI library and SHAP, similar conclusions can be made: the greatest influence on the decision made by the model for this example was that the person was male. This was followed by the influence that the person was traveling 3rd class.

Care must be taken when interpreting this graph as the input values of the black-box model, were scaled.

1.3.5 Dataset transformation

The RuleXAI library can also be used to transform a dataset. Often datasets contain missing values and nominal values. Most available algorithms do not support either missing values or nominal values. Many algorithms require the data to be rescaled beforehand. The RuleXAI library is able to convert a dataset with nominal and missing values into a binary dataset containing as attributes the conditions describing the dataset and as values "1" when the condition is satisfied for the example and "0" when the condition is not satisfied.

The data used in this notebook comes from https://sci2s.ugr.es/keel/missing.php?order=mis#sub2. It is an Australian dataset that has 14 attributes: 8 numeric and 6 nominal and 690 examples. 70% of this dataset are missing values. The attributes of this dataset are described below.

@relation australian+MV
@attribute A1 {0, 1}
@attribute A2 real[16.0,8025.0]
@attribute A3 real[0.0,26335.0]
@attribute A4 {1, 2, 3}
@attribute A5 integer[1,14]
@attribute A6 integer[1,9]
@attribute A7 real[0.0,14415.0]

@attribute A8 {0, 1}
@attribute A9 {0, 1}
@attribute A10 integer[0,67]
@attribute A11 {0, 1}
@attribute A12 {1, 2, 3}
@attribute A13 integer[0,2000]
@attribute A14 integer[1,100001]
@attribute Class {0,1}
@inputs A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14
@output Class
@data

Data load

[1]:
```
import pandas as pd
import numpy as np

train_df = pd.read_csv("./data/australian_train.csv")
test_df = pd.read_csv("./data/australian_test.csv")

train_df[["A1","A4", "A8", "A9", "A11", "A12", "Class"]] = train_df[["A1","A4", "A8", "A9
↪", "A11", "A12", "Class"]].astype(str)
test_df[["A1","A4", "A8", "A9", "A11", "A12", "Class"]] = test_df[["A1","A4", "A8", "A9",
↪ "A11", "A12", "Class"]].astype(str)

for column in train_df.select_dtypes('object').columns.tolist():
    train_df[column] = train_df[column].apply(lambda x: x.split(".")[0]).replace({"nan":␣
↪None})
    test_df[column] = test_df[column].apply(lambda x: x.split(".")[0]).replace({"nan":␣
↪None})
```

[2]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 621 entries, 0 to 620
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      559 non-null    object
 1   A2      569 non-null    float64
 2   A3      554 non-null    float64
 3   A4      541 non-null    object
 4   A5      568 non-null    float64
 5   A6      556 non-null    float64
 6   A7      559 non-null    float64
 7   A8      560 non-null    object
 8   A9      567 non-null    object
 9   A10     563 non-null    float64
 10  A11     561 non-null    object
 11  A12     549 non-null    object
 12  A13     558 non-null    float64
```

(continues on next page)

(continued from previous page)

```
 13  A14     561 non-null    float64
 14  Class   621 non-null    object
dtypes: float64(8), object(7)
memory usage: 72.9+ KB
```

[3]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69 entries, 0 to 68
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      69 non-null     object
 1   A2      69 non-null     float64
 2   A3      69 non-null     float64
 3   A4      69 non-null     object
 4   A5      69 non-null     float64
 5   A6      69 non-null     float64
 6   A7      69 non-null     float64
 7   A8      69 non-null     object
 8   A9      69 non-null     object
 9   A10     69 non-null     float64
 10  A11     69 non-null     object
 11  A12     69 non-null     object
 12  A13     69 non-null     float64
 13  A14     69 non-null     float64
 14  Class   69 non-null     object
dtypes: float64(8), object(7)
memory usage: 8.2+ KB
```

[4]:
```
train_org = train_df.copy()
test_org = test_df.copy()
```

Data preprocessing

- original data

[5]:
```
train_df.head(5)
```

[5]:
```
     A1      A2     A3 A4    A5   A6     A7 A8    A9   A10 A11   A12    A13  \
0     0  2958.0  175.0  1   4.0  4.0  125.0  0  None   0.0   1     2  280.0
1     0     NaN  115.0  1   5.0  3.0    0.0  1     1  11.0   1  None    0.0
2     1  2017.0  817.0  2   6.0  4.0  196.0  1     1   NaN   0     2   60.0
3     1  1742.0   65.0  2   3.0  4.0  125.0  0  None   0.0   0     2    NaN
4  None  5867.0  446.0  2  11.0  8.0  304.0  1     1   6.0   0     2   43.0

     A14 Class
0    1.0     0
1    1.0     1
2  159.0     1
3  101.0     0
4  561.0     1
```

- imputation of missing values

[6]:
```
cateogry_columns=train_df.select_dtypes('object').columns.tolist()
number_columns=train_df.select_dtypes('number').columns.tolist()

for column in train_df:
    if train_df[column].isnull().any():
        if(column in cateogry_columns):
            train_df[column].fillna(train_df[column].mode()[0], inplace=True)
        else:
            train_df[column].fillna(train_df[column].mean(), inplace=True)
```

[7]:
```
train_df.head(5)
```

[7]:
```
   A1           A2     A3  A4    A5   A6     A7  A8  A9       A10  A11  A12  \
0   0  2958.000000  175.0   1   4.0  4.0  125.0   0   0   0.00000    1    2
1   0  2693.896309  115.0   1   5.0  3.0    0.0   1   1  11.00000    1    2
2   1  2017.000000  817.0   2   6.0  4.0  196.0   1   1   2.49556    0    2
3   1  1742.000000   65.0   2   3.0  4.0  125.0   0   0   0.00000    0    2
4   1  5867.000000  446.0   2  11.0  8.0  304.0   1   1   6.00000    0    2

          A13    A14 Class
0  280.000000    1.0     0
1    0.000000    1.0     1
2   60.000000  159.0     1
3  185.802867  101.0     0
4   43.000000  561.0     1
```

- one hot encoding

[ ]:
```
data = pd.concat([train_df, test_df], axis = 0)
data.reset_index(drop=True,inplace=True)
data_with_dummies = pd.get_dummies(data.drop(["Class"], axis=1))

train_df_encoded =  data_with_dummies[:train_df.shape[0]]
train_df_encoded["Class"] = data[:train_df.shape[0]]["Class"]

test_df_encoded =  data_with_dummies[train_df.shape[0]:]
test_df_encoded["Class"] = data[train_df.shape[0]:]["Class"]
```

[9]:
```
train_df_encoded.head(5)
```

[9]:
```
            A2     A3    A5   A6     A7       A10         A13    A14  A1_0  \
0  2958.000000  175.0   4.0  4.0  125.0   0.00000  280.000000    1.0     1
1  2693.896309  115.0   5.0  3.0    0.0  11.00000    0.000000    1.0     1
2  2017.000000  817.0   6.0  4.0  196.0   2.49556   60.000000  159.0     0
3  1742.000000   65.0   3.0  4.0  125.0   0.00000  185.802867  101.0     0
4  5867.000000  446.0  11.0  8.0  304.0   6.00000   43.000000  561.0     0

   A1_1  ...  A8_0  A8_1  A9_0  A9_1  A11_0  A11_1  A12_1  A12_2  A12_3  Class
0     0  ...     1     0     1     0      0      1      0      1      0      0
1     0  ...     0     1     0     1      0      1      0      1      0      1
2     1  ...     0     1     0     1      1      0      0      1      0      1
3     1  ...     1     0     1     0      1      0      0      1      0      0
```

(continues on next page)

(continued from previous page)

```
4      1  ...      0      1      0      1       1       0       0       1       0       1

[5 rows x 23 columns]
```

- normalization

[10]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

train_df_encoded_and_scaled = train_df_encoded.copy()
train_df_encoded_and_scaled[['A2','A3','A5','A6', 'A7', 'A10', 'A13', 'A14']] = scaler.
↪fit_transform(train_df_encoded[['A2','A3','A5','A6', 'A7', 'A10', 'A13', 'A14']])

test_df_encoded_and_scaled = test_df_encoded.copy()
test_df_encoded_and_scaled[['A2','A3','A5','A6', 'A7', 'A10', 'A13', 'A14']] = scaler.
↪transform(test_df_encoded[['A2','A3','A5','A6', 'A7', 'A10', 'A13', 'A14']])
```

[11]:
```
train_df_encoded_and_scaled.head(5)
```

[11]:
```
         A2        A3        A5        A6        A7       A10           A13  \
0  0.182967 -0.348773 -0.952571 -0.356525 -0.240176 -0.518373  5.508121e-01
1  0.000000 -0.370201 -0.667503 -0.887967 -0.331488  1.766525 -1.086471e+00
2 -0.468944 -0.119484 -0.382434 -0.356525 -0.188311  0.000000 -7.356248e-01
3 -0.659460 -0.388059 -1.237640 -0.356525 -0.240176 -0.518373  1.661943e-16
4  2.198282 -0.251986  1.042910  1.769244 -0.109417  0.727935 -8.350313e-01

        A14  A1_0  A1_1  ...  A8_0  A8_1  A9_0  A9_1  A11_0  A11_1  A12_1  \
0 -0.196556     1     0  ...     1     0     1     0      0      1      0
1 -0.196556     1     0  ...     0     1     0     1      0      1      0
2 -0.166737     0     1  ...     0     1     0     1      1      0      0
3 -0.177684     0     1  ...     1     0     1     0      1      0      0
4 -0.090868     0     1  ...     0     1     0     1      1      0      0

   A12_2  A12_3  Class
0      1      0      0
1      1      0      1
2      1      0      1
3      1      0      0
4      1      0      1

[5 rows x 23 columns]
```

[12]:
```
X_train = train_df_encoded_and_scaled.drop(columns = "Class")
y_train =train_df_encoded_and_scaled["Class"]

X_test = test_df_encoded_and_scaled.drop(columns = "Class")
y_test = test_df_encoded_and_scaled["Class"]
```

Building a Random Forest model on a preprocessed dataset

[13]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import balanced_accuracy_score

clf = RandomForestClassifier(random_state=42)

clf.fit(X_train, y_train)
```

[13]:
```
RandomForestClassifier(random_state=42)
```

Balanced accuracy on training set

[14]:
```
balanced_accuracy_score(y_train,clf.predict(X_train))
```

[14]:
```
1.0
```

Balanced accuracy on test set

[15]:
```
balanced_accuracy_score(y_test,clf.predict(X_test))
```

[15]:
```
0.8153846153846154
```

Using RuleXAI to transform the original set

[16]:
```
X_train_org = train_org.drop(columns = "Class")
y_train_org = train_org["Class"]

X_test_org = test_org.drop(columns = "Class")
y_test_org = test_org["Class"]
```

[17]:
```
X_train_org.head(5)
```

[17]:
```
     A1      A2     A3 A4    A5   A6     A7 A8    A9   A10 A11   A12    A13  \
0     0  2958.0  175.0  1   4.0  4.0  125.0  0  None   0.0   1     2  280.0
1     0     NaN  115.0  1   5.0  3.0    0.0  1     1  11.0   1  None    0.0
2     1  2017.0  817.0  2   6.0  4.0  196.0  1     1   NaN   0     2   60.0
3     1  1742.0   65.0  2   3.0  4.0  125.0  0  None   0.0   0     2    NaN
4  None  5867.0  446.0  2  11.0  8.0  304.0  1     1   6.0   0     2   43.0

     A14
0    1.0
1    1.0
2  159.0
3  101.0
4  561.0
```

[18]:
```
from rulexai.explainer import Explainer

explainer =  Explainer(X = X_train_org,model_predictions = y_train_org, type =
↪"classification").explain()
```

```
[19]: X_train_tranformed = explainer.fit_transform(X_train_org, selector=None)
```

```
[20]: X_train_tranformed.head(5)
```

```
[20]:    A2 = <19.0, 7037.5)  A8 = {0}  A10 = (-inf, 10.5)  A13 = (-inf, 216.0)  \
0                     1         1                    1                     0
1                     0         0                    0                     1
2                     1         0                    0                     1
3                     1         1                    1                     0
4                     1         0                    1                     1

   A5 = (-inf, 1.5)  A2 = <2445.5, 4429.0)  A5 = (-inf, 3.5)  A9 = {0}  \
0                 0                      1                 0         0
1                 0                      0                 0         0
2                 0                      0                 0         0
3                 0                      0                 1         0
4                 0                      0                 0         0

   A2 = <1816.5, 3779.0)  A13 = <110.0, inf)  ...  A6 = <2.0, inf)  \
0                      1                   1  ...                1
1                      0                   0  ...                1
2                      1                   0  ...                1
3                      0                   0  ...                1
4                      0                   0  ...                1

   A7 = <168.0, inf)  A2 = <29.5, inf)  A3 = (-inf, 12.5)  A5 = <7.5, inf)  \
0                  0                 1                  0                0
1                  0                 0                  0                0
2                  1                 1                  0                0
3                  0                 1                  0                0
4                  1                 1                  0                1

   A14 = (-inf, 1069.5)  A3 = (-inf, 1080.0)  A5 = <6.5, inf)  \
0                     1                    1                0
1                     1                    1                0
2                     1                    1                0
3                     1                    1                0
4                     1                    1                1

   A13 = (-inf, 591.5)  A6 = <3.5, inf)
0                    1                1
1                    1                0
2                    1                1
3                    0                1
4                    1                1

[5 rows x 99 columns]
```

Building a Random Forest model on a prepared dataset by RuleXAI

[21]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=42)

clf.fit(X_train_tranformed, y_train_org)
```

[21]:
```
RandomForestClassifier(random_state=42)
```

[22]:
```
X_test_transformed = explainer.transform(X_test_org)
```

Balanced accuracy on training set

[23]:
```
balanced_accuracy_score(y_train_org,clf.predict(X_train_tranformed))
```

[23]:
```
1.0
```

Balanced accuracy on test set

[24]:
```
balanced_accuracy_score(y_test_org,clf.predict(X_test_transformed))
```

[24]:
```
0.844871794871795
```

Comparing the results obtained with RandomForest on the preprocessed original set (imputation, dummification, normalization) and on the original set transformed with RuleXAI, it can be seen that these results are similar.

INDEX