Pontifícia Universidade Católica
do Rio de Janeiro

INF2102: Interface for Optimal Counterfactual Explanations in Tree Ensembles

IT Department

Moises H. Pereira
mpereira@inf.puc-rio.br

List of Figures

1 ViCE interface 5
2 VIDA Lab project full screen 6
3 VIDA Lab project user data point screen 6
4 General use case diagram 9
5 The initial user interface 11
6 Component used by binary features 12
7 Component used by numerical features 12
8 Component used by categorical features 13
9 Main user interface after generating the counterfactual explanation 13
10 Some tests over the Controller component 14
11 Pytest result 15
12 COMPAS dataset example 15

Summary

1 Introduction **4**

2 Related works **4**

3 Specification **7**

3.1 Back-end Requirements 7

3.1.1 Functional Requirements 7

3.1.2 Non-functional Requirements 8

3.2 Front-end Requirements 8

3.2.1 Functional Requirements 8

3.2.2 Non-functional Requirements 9

3.3 Use case 9

4 Architecture and Project **10**

4.1 Model 11

4.2 View 11

4.3 Controller 12

5 Tests **14**

6 User Guide **15**

References **17**

1 Introduction

Machine learning (ML) algorithms are incredibly present in our daily, and their spread is increasing, from film and music recommendation systems to high-risk areas as healthcare, criminal justice, finances, and so on, helping the decision-taking. But the complexity of building the ML algorithms is increasing too, while their interpretability is decreasing. Thus, the algorithms and their decisions cannot be easily explained by developers or users, and the algorithms cannot explain themselves. So mistakes and biases could not be detected, and this can profoundly impact human lives (Carvalho, Pereira, & Cardoso, 2019).

Because of this, the theme of transparency and explainability is growing. This need for explanation can also be seen in the new regulation about personal data protection and treatment, approved in 2016 for the European Union (Goodman & Flaxman, 2017). So to obtain the explanations, the researchers are studying solutions, and a potential approach to this problem is called Counterfactual Explanation.

The main idea in a counterfactual explanation is to answer the question: "What is the minimum change that input needs to achieve the desired output?". For example: "What do I need to change in my data to get a loan previously denied by a bank?" (Artelt & Hammer, 2019).

Another example is the following: someone wants to rent out your apartment charging over 1000 euros, but the maximum rent that can be achieved is 900 euros. So, what apartment features can be changed to achieve the desired rent value? A potential counterfactual explanation can say that if the apartment becomes $15m^2$ larger, the rent can increase to 1000 euros, but this is a problem because an apartment cannot have its size increased (not actionable). Another counterfactual explanation can suggest allowing pets and upgrading the insulation, which is possible to do and easy to understand (Molnar, 2020).

So, to face this need, the "Optimal Counterfactual Explanations in Tree Ensembles" (OCEAN)work was done. This work can generate counterfactual explanations using Mixed Integer Programing and Isolation Forest to ensure plausibility. Besides that, this work contributes to building efficient mathematical models over Tree Ensembles; also, the Isolation Forest gives a view that prevents outliers explanations, guaranteeing plausibility; and provides a code that can be adjusted to specific needs (Parmentier & Vidal, 2021).

Considering that previous work, an improvement over it can be to produce a user-friendly interface to help lay users to produce and "negotiate" a useful counterfactual explanation. So, this document presents the development of the "Interface for Optimal Counterfactual Explanations in Tree Ensembles" work. This user interface is going to use the previous work codes as API, so I will not change the codes of previous work, but I will consult the files and classes, sending the needed information, getting the answers, and intermediating the negotiations said above.

2 Related works

To base this work, we use two main references (Gomez, Holter, Yuan, & Bertini, 2020) and (Piesanen & Kerrigan, 2020), those two projects present two different solutions to counterfactual explanation visualization. Then, we studied both of them, and we take the features that are interesting to the present work.

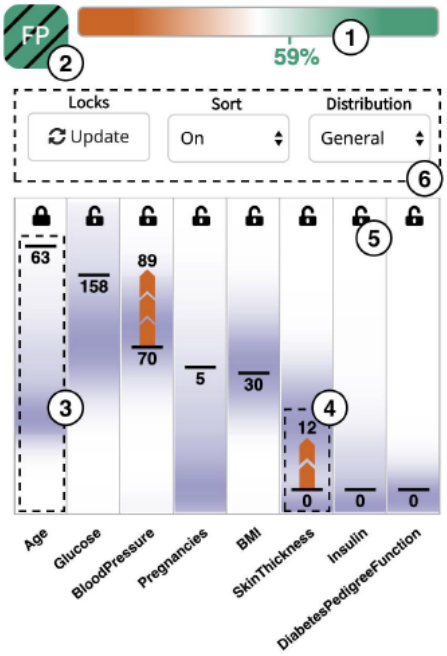

Figure 1: ViCE interface

From (Gomez et al., 2020) Figure 1, we consider the idea of the bars to show the features and their respective values, and the idea to lock and unlock the features to represent the actionability constraint.

From (Piesanen & Kerrigan, 2020) Figure 2 and Figure 3, we also consider the way that the authors use to set the actionability constraints, and the way for the user to select a data point from the dataset or to set his own values. Besides that, some more references have been used to ground this work, like (Code, 2021), (Cheng, Ming, & Qu, 2020), (Kahng, Andrews, Kalro, & Chau, 2017), (Kulesza, Burnett, Wong, & Stumpf, 2015) and more.

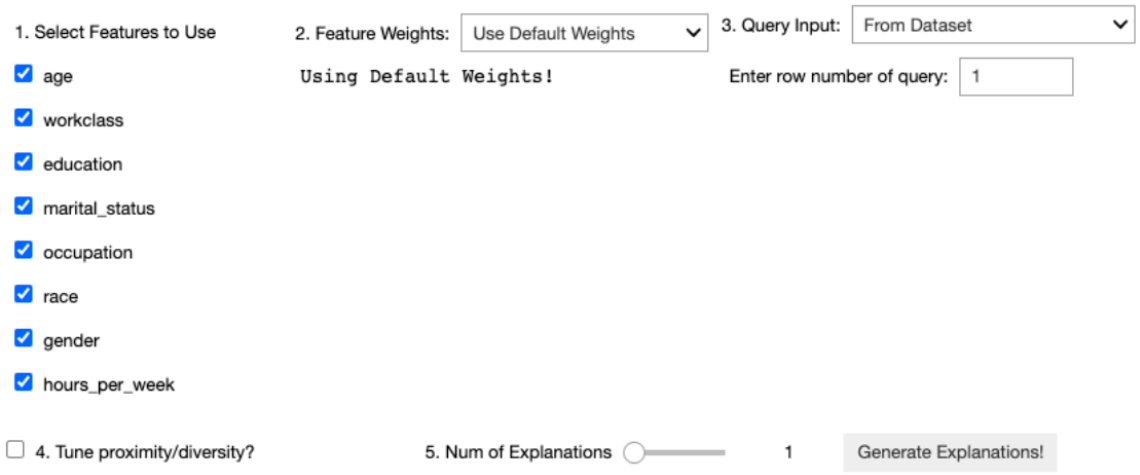

Figure 2: VIDA Lab project full screen

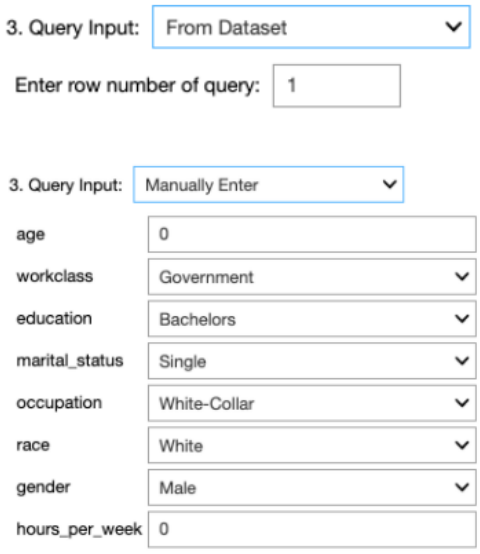

Figure 3: VIDA Lab project user data point screen

3 Specification

3.1 Back-end Requirements

3.1.1 Functional Requirements

FR 1: initialize interface

- Description: the system initializes the interface considering the default parameters and instantiating the needed components
- Input: start event
- Process: the system verifies the available datasets to fill the selection dataset component, instantiate the needed components to show the features information, and instantiate the whole interface
- Output: interface

FR 2: random point selection

- Description: the system selects a random value in the dataset range index
- Input: a request for a random data point
- Process: the system takes the selected dataset, analyses its length, and selects a random value considering it
- Output: the selected random data point

FR 3: calculate class

- Description: the system receives a data point and uses the trained model to predict the corresponding class
- Input: a data point
- Process: the system uses the Random Forest trained model to predict the class given a user data point
- Output: predicted class

FR 4: calculate counterfactual explanation

- Description: the system receives a data point and returns a corresponding counterfactual explanation
- Input: a data point
- Process: the system takes the input data point and calculates the counterfactual explanation
- Output: a data point that means the counterfactual explanation

3.1.2 Non-functional Requirements

NRF 1: language: Python
NRF 2: main libraries: sklearn, gurobipy, pyTest

3.2 Front-end Requirements

3.2.1 Functional Requirements

FR 1: dataset selection

- Description: the user selects a dataset given a list of datasets
- Input: a selection event
- Process: given a list of datasets name the user selects one of them to be open in the back-end
- Output: the system shows a list of features informing the minimum and maximum value when the feature is numerical, displays a list with the possible values when the feature is categorical and shows the two possible values when the feature is binary

FR 2: random point selection

- Description: the user selects an option to receive a random data point from the selected dataset
- Input: a selection event
- Process: when the user selects this option, the system will request a data point from the selected dataset taking a random index
- Output: the data point requested

FR 3: class calculation

- Description: the user selects an option to calculate the class for the selected data point
- Input: a selection event
- Process: when the user selects this option, the system will calculate the class for the selected data point values
- Output: the class

FR 4: counterfactual generation

- Description: the user selects an option to calculate the counterfactual explanation for the selected data point values
- Input: a selection event
- Process: when the user selects this option, the system will generate the counterfactual explanation considering the current value and the constraints over each feature
- Output: a comparison between the original data point values and the counterfactual explanation data point values

3.2.2 Non-functional Requirements

NFR 1: platform: the software will be developed for Desktop
NFR 2: language: Python
NFR 3: library: pyQt
NFR 4: update minimum, maximum, and current value for numerical features

- Description: the user edit the thresholds and the actual value to the counterfactual explanation

NFR 5: update allowed, not allowed, and current value for categorical features

- Description: the user indicates what feature values is considered allowed and not allowed to the counterfactual explanation

3.3 Use case

The system has just one use case (Figure 4), and over it, the user can execute all the system functionalities.

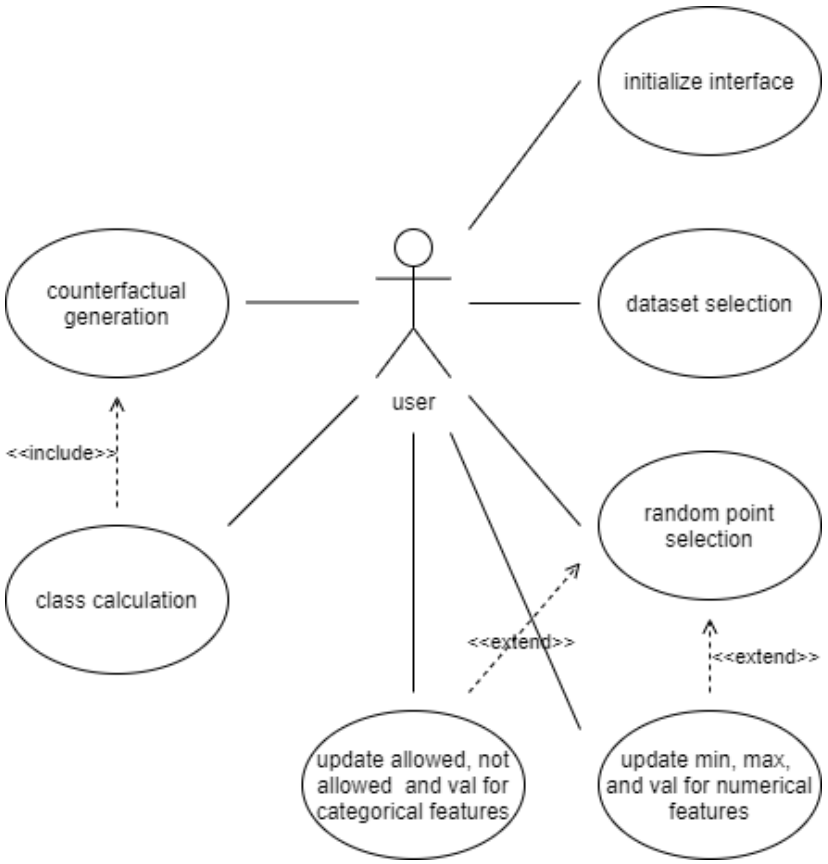

Figure 4: General use case diagram

Flow:

1. user opens the interface
2. user selects a dataset
3. user chooses a random data point or fills in all the values
4. user can impose some additional constraints
5. user selects the option “Calculate Class” and/or “Generate Counterfactual”
6. the system shows the result
7. user can return to step 3 to try another data point or to step 4 to try new constraints

4 Architecture and Project

We decided to use an architecture based on Model-View-Controller, where the View will be used to show the interface and handle some getters and setters to interact with the user; the Model will be used to group some operations over the dataset; the Controller will be used to manage the View and the Model, besides that the controller will be responsible for accessing the Engine functions. The details of each component will be well explained later.

There is the ui folder that groups the whole project, inside it there are five folders: CounterfactualEngine, CounterfactualInterface, MainApplication, Pytest, and Resources. Each of those folders group some files and the CounterfactualInterface folder also group some folders:

- CounterfactualEngine: CounterfactualEngine files
- CounterfactualInterface: each of the following folders, group the subcomponent files
  - folders: ComboboxList, DoubleRadioButton, LineEditMinimumMaximum, and Slider3Ranges
  - files: model, view, controller, worker, enum, and the ui file
- MainApplication: MainApplication files
- Pytest: test files
- Resources: use case file

The file names have been standardized as the same name of the class inside them; all the test files start with the prefix “test_”; and the names of the attributes have been standardized as follow:

- $'\_\_attrName'$ if the attribute is private
- $'\_attrName'$ if the attribute is protected
- $'attrName'$ if the attribute is public

4.1 Model

The Model component handles dataframe and data point operations, and access some OCEAN codes to open the desired dataset and get some features information to generate the counterfactual explanation.

Talking about the data point operations, it is important to know that these operations handle to:

- get a random data point
- transform a data point to be used to predict its class and calculate the counterfactual; this transformation consists of applying a min-max scaling over the values of the numerical features and appending all possible values of the categorical features considering them as binary features
- apply min-max scale to a single value to append the new constraints
- invert the transformation mentioned above

4.2 View

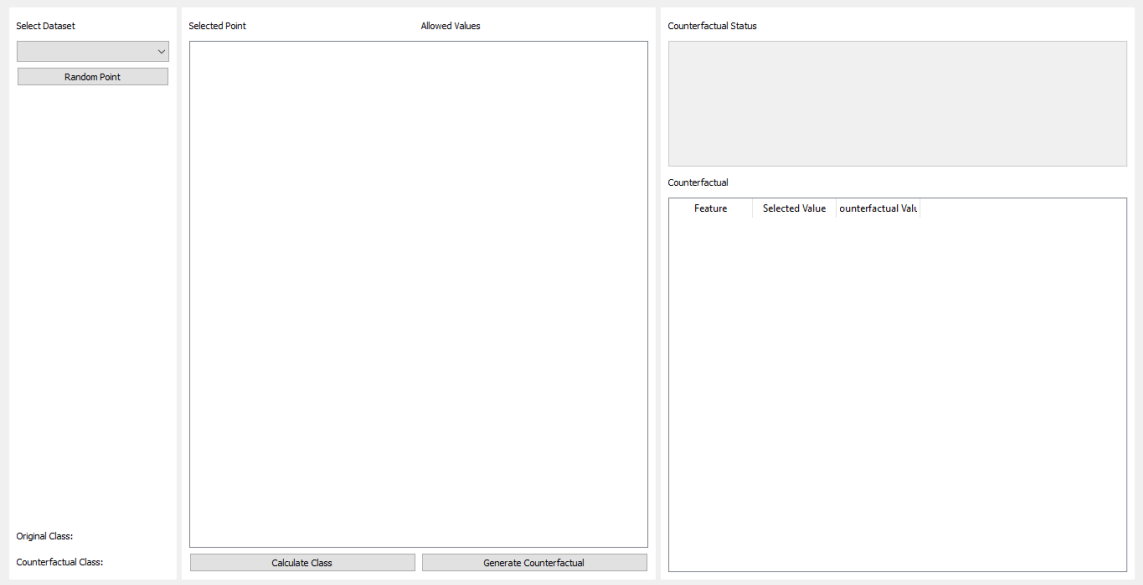

Figure 5: The initial user interface

The View component is the way that the user interacts with the system selecting the dataset, the desired data point, pressing the button Calculate class, and the button Generate Counterfactual. Also, the user can insert new constraints over the data point using the subcomponents below.

The first subcomponent (Figure 6) is used to show binary features; it is the simplest subcomponent because just needs to show the two possible values to the user to select.

The second subcomponent (Figure 7) is used to show numerical features; on the left side, the user can increase or decrease the minimum bound, and on the right side, the user can also increase or decrease the maximum bound, besides that, in the middle, the user can drag the ranges to specify the

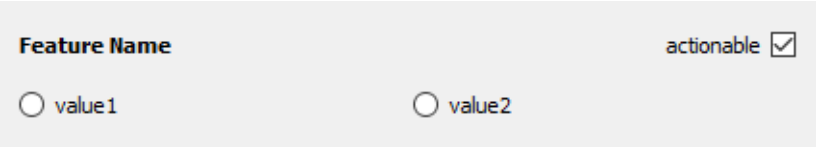

Figure 6: Component used by binary features

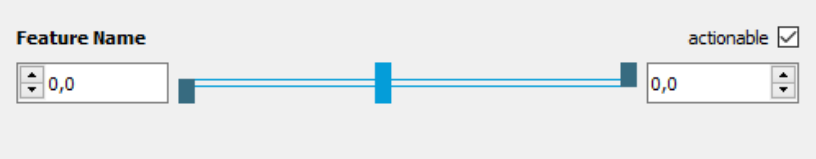

Figure 7: Component used by numerical features

minimum and maximum allowed values, and specify the current value to the feature. The minimum and maximum allowed values explicit new constraints to the optimization model.

The third component (Figure 8) is used to show categorial features; on the left side, the user can select an option that corresponds with his own value inside the combobox, and on the right side, the user can see all possible values for that feature and deselect the values that are not allowed. The not allowed values are considered as new constraints to the system.

These three components also give the user the option to inform that the corresponding feature is not actionable; in other words, the feature value informed for the user cannot change when generating the counterfactual explanation.

Considering the full use of the system, the user will see Figure 9. Once the counterfactual explanation is done and the user saw it, he can use the subcomponents above to add more specific constraints and obtain a personal counterfactual explanation.

4.3 Controller

The Controller compoenent is responsible to capture the interactions over the View and handle them properly; instantiate and manage the Model, calling the functions needed to fill the View; get the answers and update the View; instantiate the worker class to access the counterfactual explanation generator.

The worker class is like a subpart of the Controller, he also has access to the Model to get the features information, and from the Controller, the worker can get the selected data point; the worker also is able to apply all the user constraints.

To do worker class function properly is needed to have a random forest classifier and optionally an isolation forest, so, to train both models we use an engine class, and after that this class can be used to do predictions about data points.

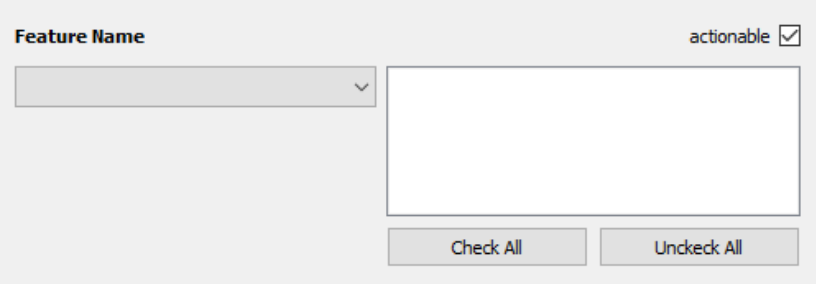

Figure 8: Component used by categorical features

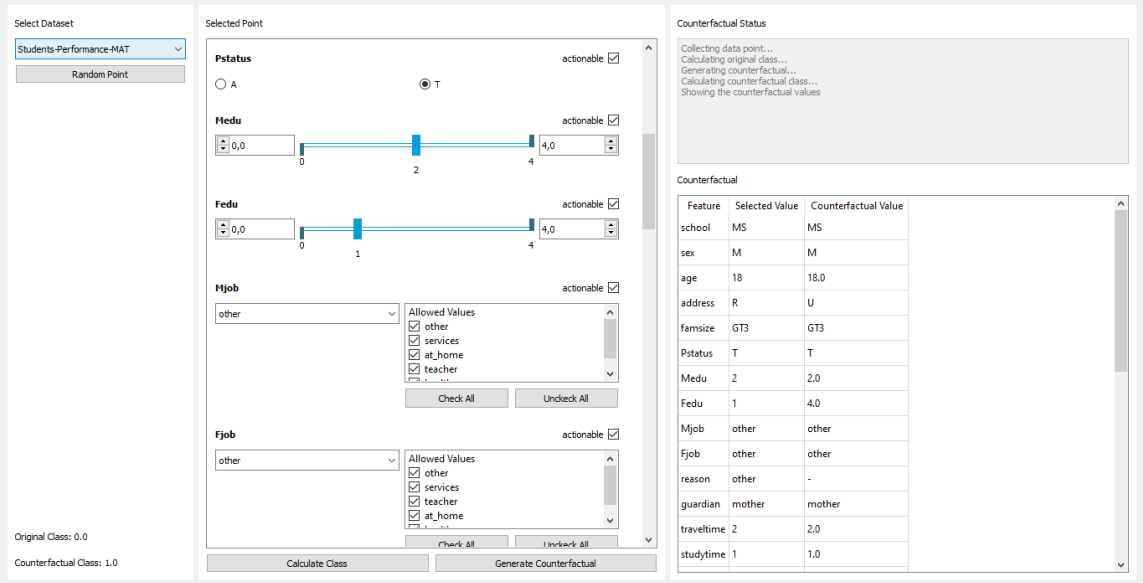

Figure 9: Main user interface after generating the counterfactual explanation

5 Tests

We decided to use the library pytest to automate the test task. It is worth mentioning that the tests are unitary tests over the classes that I implemented, the OCEAN code was considered like a black-box where I used as an API. Then, the tests are input and output tests over the Model, View, Controller, Worker, and the subcomponents classes.

```
# the function reportProgress expect a string as parameter
# send another type would arrise an assertionError
@pytest.mark.parametrize('progress', [1, 2.9, False, ('t1', 't2'), [], None])
def test_CIC_reportProgress_wrong_type_parameter(progress):
    with pytest.raises(AssertionError):
        app = QtWidgets.QApplication(sys.argv)
        counterfactualInterfaceController = StaticObjects.staticCounterfactualInterfaceController()
        counterfactualInterfaceController.reportProgress(progress)

# the function reportProgress expect a string as parameter
# send it would not arrise assertionError
def test_CIC_reportProgress_right_parameter():
    app = QtWidgets.QApplication(sys.argv)
    counterfactualInterfaceController = StaticObjects.staticCounterfactualInterfaceController()
    counterfactualInterfaceController.reportProgress('progress: OK')

# the function updateCounterfactualClass expect a not none parameter
# send none would arrise an assertionError
def test_CIC_updateCounterfactualClass_none_parameter():
    with pytest.raises(AssertionError):
        app = QtWidgets.QApplication(sys.argv)
        counterfactualInterfaceController = StaticObjects.staticCounterfactualInterfaceController()
        counterfactualInterfaceController.updateCounterfactualClass(None)

# the function updateCounterfactualClass expect a not none parameter
# send a different content would not arrise an assertionError
@pytest.mark.parametrize('counterfactualClass', [1, [1], (1), [(1)]])
def test_CIC_updateCounterfactualClass_right_parameter(counterfactualClass):
    app = QtWidgets.QApplication(sys.argv)
    counterfactualInterfaceController = StaticObjects.staticCounterfactualInterfaceController()
    counterfactualInterfaceController.updateCounterfactualClass(counterfactualClass)
```

Figure 10: Some tests over the Controller component

Figure 10 shows how the tests are implemented, some tests use wrong parameters to arise the expected assertion errors (these errors are caught by "with pytest.raises(AssertionError)"), and other ones use the right parameters to verify if everything is ok. To test possibilities over a single function, we use the option "@pytest.mark.parametrize(parameterName, [values])".

To run all tests it is necessary to use the command "pytest" inside the ui folder. The results are shown in Figure 11. Besides that, it is possible to run the tests showing more information, to do it, it is necessary to use the command "pytest -v", also, inside the ui folder.

Figure 11: Pytest result

6 User Guide

What is the required format for the dataset? (Even if this definition is not part of your system *per se*, it's a prerequisite for its usage, so it would be nice to have some information about it.)

The system already has eight datasets (Adult, Credit Card Default, COMPAS, German Credit, Online News, Data Phishing, Spambase, Students Performance) to be used, but it is simple to add new ones. For the system to work properly, it needs some information associated with those features to tell about the feature type and actionability. The feature types are binary (the set of feature values has just two possible values), categorical (the set of feature values has more than two values), discrete (integer numbers), and numerical (continuous numbers), so, to inform it, we use the abbreviations B, C, D, and N respectively. The feature actionabilities are free (the feature value can change without constraints), fixed (the feature value cannot change), increasing (the feature value just can be increased), and predict (inform that the feature value is the class), so, to inform it, we use the abbreviations FREE, FIXED, INC, and PREDICT respectively.

```
AgeGroup,Race,Sex,PriorsCount,ChargeDegree,Class
D,B,B,D,B,B
INC,FREE,FIXED,FREE,FREE,PREDICT
```

Figure 12: COMPAS dataset example

Consider the COMPAS dataset (Figure 12) as an example, and it has the following feature names and the necessary information.

To run the system, the user needs to have the gurobi solver and a license to it (free academic licenses are available), besides that, the user needs to install all the dependencies in the requirements

file using the command: “pip install uiRequirements.txt”.

After that, the user just needs to run the main file inside the ui folder, using the command: “python main.py”, and interact with the system interface as explained in the sections above.

References

Artelt, A., & Hammer, B. (2019). Efficient computation of counterfactual explanations of lvq models. *arXiv preprint arXiv:1908.00735*.

Carvalho, D. V., Pereira, E. M., & Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, *8*(8), 832.

Cheng, F., Ming, Y., & Qu, H. (2020). Dece: Decision explorer with counterfactual explanations for machine learning models. *IEEE Transactions on Visualization and Computer Graphics*.

Code, P. (2021). *What-if tool.* `https://pair-code.github.io/what-if-tool/`.

Gomez, O., Holter, S., Yuan, J., & Bertini, E. (2020). Vice: visual counterfactual explanations for machine learning models. In *Proceedings of the 25th international conference on intelligent user interfaces* (pp. 531–535).

Goodman, B., & Flaxman, S. (2017, October). European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, *38*(3), 50–57. doi: 10.1609/aimag.v38i3.2741

Kahng, M., Andrews, P. Y., Kalro, A., & Chau, D. H. (2017). A cti v is: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, *24*(1), 88–97.

Kulesza, T., Burnett, M., Wong, W.-K., & Stumpf, S. (2015). Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th international conference on intelligent user interfaces* (pp. 126–137).

Molnar, C. (2020). *Interpretable machine learning.* Lulu. com.

Parmentier, A., & Vidal, T. (2021). *Optimal counterfactual explanations in tree ensembles.*

Piesanen, & Kerrigan. (2020). *Vida lab.* `https://sites.google.com/nyu.edu/counterfactual-explorer/`.