Integrating PyMOO with DiCEML

Md Asif Tanvir
Department of Computer Science
Missouri State University
Springfield, Missouri, USA
mt5864s@missouristate.edu

Md Abdur Rahman Fahad
Department of Computer Science
Missouri State University
Springfield, Missouri, USA
mf8494s@missouristate.edu

***Abstract*—Counterfactual explanations help us understand machine learning models. They show how changing inputs can change predictions. DiCEML is a popular tool for generating such explanations. It uses methods like random sampling, genetic algorithms, and KD-trees. In this study, we improved DiCEML by adding PyMOO. PyMOO is a Python library for solving problems with multiple goals. This addition makes counterfactual generation better. It helps balance goals like keeping changes small, making explanations diverse, and ensuring they are realistic. We used PyMOO's algorithms, such as NSGA-II, to achieve this balance. These algorithms create explanations that work well for different needs. This paper explains how we combined DiCEML and PyMOO. It describes the steps we followed and the results we achieved. Our early results show that this method makes AI systems clearer. It also helps users understand how predictions are made. This work aims to make machine learning easier to use and trust.**

***Index Terms*—Counterfactuals, Multi-Objective Optimization, DiCEML, PyMOO**

I. Introduction

Machine learning (ML) models are being used more and more in important areas like healthcare, finance, and smart homes. It is important to make these models easier to understand. People often want to know not just what the model predicts but why it made that prediction. Counterfactual explanations help answer these questions by showing how changing certain inputs could lead to a different prediction.

DiCE (Diverse Counterfactual Explanations) [1] is a tool that helps create explanations for many existing machine learning models. It aims to make these explanations simple, varied, and useful. However, the methods DiCE uses now, like random sampling, genetic algorithms, and KD-trees, are not very effective when we need to balance several goals at once. For example, users might want explanations that change inputs only a little, stay different from each other, and still keep the predictions accurate.

Our research question is about creating better counterfactual explanations by combining different tools. These explanations should balance accuracy, ease of understanding, and practicality. This can make machine learning models easier to understand and use.

Our research tries to address this question by using multi-objective optimization. This approach creates diverse explanations that are all equally good in different ways. This gives users more choices to pick explanations that work best for them. We think using PyMOO's [2] algorithms inside DiCE ML can help create better and more varied counterfactual explanations.

In this study we integrated DiCEML [1] with PyMOO [2], a library that specializes in multi-objective optimization. PyMOO uses algorithms like NSGA-II, MOEA/D to create explanations that balance different goals, such as making small, meaningful changes and keeping explanations diverse. This integration helps generate counterfactual explanations by balancing multiple objectives, such as accuracy, interpretability, and feasibility. These improvements aim to make machine learning models more transparent and usable.

This paper describes how we modified DiCEML to work with PyMOO, the steps we followed, and the results we achieved. It also highlights how multi-objective optimization can produce a set of diverse, Pareto-optimal counterfactuals, allowing users to choose explanations that best suit their needs.

The rest of the paper is organized as follows: Section II reviews related work and discusses the strengths and weaknesses of current methods for generating counterfactuals. Section III explains how we combined DiCEML with PyMOO, including the changes we made. Section IV shares the results of our experiments. Section V and VI talks about the challenges we faced and future plans. Finally, Section VII summarizes what we learned from this study.

II. Literature Review

Counterfactual explanations are a powerful tool to explain predictions made by machine learning models. The paper [1] addresses the need for explainability in machine learning by introducing diverse counterfactual explanations. It focuses on generating explanations that allow users to understand model behavior through what-if scenarios, emphasizing diversity in the explanations provided. The authors propose a framework that generates multiple counterfactual instances by looking at four main factors Diversity, Proximity, Sparsity and last but not the least User defined constraints on features. The authors used three main strategies to sample counterfactuals, which are Random Sampling, Genetic Algorithm and KdTree. As Evaluation metrics the authors used Validity, Proximity, Sparsity and Diversity.

Here we dive deep into the DiCE library. The DiCE library helps generate ”what-if” examples, which are called counterfactuals. Counterfactuals answer questions like:

Given that the model's output for input x is y, what changes to x would result in a desired output y^*?

For example, if x is the input, $f(x)$ is the model's output, and y^* is the desired output, we solve:

$$\text{Find } x^* \text{ such that } f(x^*) = y^*, \text{ with } \|x - x^*\| \text{ minimized.} \tag{1}$$

This means we aim to find a new input x^* that changes the output to y^* while keeping changes to x as small as possible. Counterfactuals should also be diverse and realistic. For instance, unrealistic changes (e.g., reducing a personś age from 30 to 20) are less useful. DiCE allows setting limits for features using the permitted range parameter to ensure feasibility.

Counterfactuals also explain necessity and sufficiency. A feature value x_i is **necessary** for the output y if changing x_i changes y, while keeping all other features fixed. Mathematically:

$$\text{If } f(x) \neq f(x_{\neg i}), \text{ where } x_{\neg i} \text{ is } x \text{ with } x_i \text{ changed.} \tag{2}$$

A feature value x_i is **sufficient** if y cannot change when x_i is fixed. DiCE uses the `features_to_vary` parameter to test these conditions.

DiCE generates counterfactuals using two methods:

- **Model-Agnostic Methods:** These work for any ML model, including black-box models. They sample points near x and optimize for proximity, diversity, and feasibility. Examples include:
  - Randomized Search
  - Genetic Search
  - KD Tree Search
- **Gradient-Based Methods:** These require differentiable models (e.g., neural networks). They use gradient descent to minimize a loss function that considers proximity and diversity.

For feature importance, counterfactuals identify which features change most often to achieve a desired output. This local importance can be averaged across samples to find global importance. Compared to methods like LIME [4] or SHAP [5], DiCE often highlights a broader range of important features.

Another related paper [2] is about a Python library for solving multi-objective optimization problems. It provides tools for evolutionary algorithms, which are key for problems where multiple objectives need to be balanced. The paper explains the functionality of pymoo, detailing the implementation of evolutionary algorithms like $NSGA-II$ and $MOEA/D$. It highlights the library's flexibility in allowing users to customize objectives, constraints, modular implementation and distributed computation. PyMoo provides tools for solving problems with multiple conflicting objectives. PyMOO defines a general optimization problem as:

$$\begin{aligned}
&\text{Minimize } f_m(x),\, m = 1, \ldots, M,\\
&\text{Subject to } g_j(x) \leq 0,\, j = 1, \ldots, J,\\
&h_k(x) = 0,\, k = 1, \ldots, K,\\
&x_i^L \leq x_i \leq x_i^U,\, i = 1, \ldots, N,
\end{aligned}$$

where $f_m(x)$ are the objective functions, $g_j(x)$ and $h_k(x)$ are inequality and equality constraints, and x_i^L, x_i^U are variable bounds. PyMOO also supports customization of algorithms through operators like sampling, crossover, and mutation. For example, crossover combines parent solutions to produce offspring, while mutation introduces diversity.

The authors of paper [3] merges counterfactual explainability with multi-objective optimization, focusing on generating explanations that satisfy multiple criteria simultaneously, such as interpretability, proximity, and feasibility. They made their framework model-agnostic and handles classification, regression and mixed feature spaces. To generate diverse and interpretable counterfactual explanations, the authors formalized the problem as the goal to find a counterfactual x' for a given instance x^* such that the prediction $f(x')$ is close to a desired outcome Y', while balancing proximity to x^*, sparsity, and plausibility. This can be expressed as:

$$\min_{x'} o(x') = \big(o_1(f(x'), Y'), o_2(x', x^*), o_3(x', x^*), o_4(x', X^{\text{obs}})\big), \tag{3}$$

The primary metrics they used for objectives are $O1$ (Distance between Prediction and Actual Label), $O2$ (Distance between Acual Input and Counterfactuals using Gower distance), $O3$ (How many features have been changed), $O4$ (Weighted average Gower distance between actual input and the k nearest observed data points).

III. Methodology

To integrate Pymoo with DiCE we need to understand the internal workings of both the softwares. In the following subsections, we have described our proposed approach for integrating the system, the modifications made in the source codes, and the tools used to accomplish this process.

A. Proposed Approach

1) Modifying the DiCE ML Framework: To incorporate PyMoo's algorithms as a new sampling strategy in DiCE, we adapted the DiCE ML framework as follows:

- Extending the ExplainerBase Class: The `ExplainerBase` class in DiCE defines essential structures and methods required for generating counterfactuals. To integrate PyMoo, we created a custom subclass that extends the `ExplainerBase` class, allowing us to implement PyMoo as an additional sampling strategy.
- Implementing Abstract Methods: Key methods in the `ExplainerBase` class, particularly `_generate_counterfactuals()`, are designed to define the generation process of counterfactual examples. We will override the `_generate_counterfactuals()` method in our custom subclass, integrating PyMoo's multi-objective optimization algorithms to generate counterfactuals based on various objectives, such as proximity, sparsity, and diversity.

2) Integrating PyMoo as a Counterfactual Sampling Strategy: DiCE ML currently supports three sampling strategies, including Random Sampling, Genetic Algorithm, and KdTree. We need to add Pymoo as a new sampling strategy inside DiCE as described in figure 1. The following modifications need to be made to incorporate PyMoo as an additional sampling option:

- Adding PyMoo to Sampling Strategies: PyMoo will be integrated to expand DiCE's capabilities, allowing access to optimization algorithms such as NSGA-II and MOEA/D, which are effective for multi-objective optimization. These algorithms enable the generation of counterfactual that consider multiple criteria, balancing trade-offs for explanations that meet diverse user needs.
- Setting Up PyMoo Algorithms: Specific PyMoo algorithms will be selected based on their ability to balance objectives. Initially, we will only use NSGA-II.

3) Generating and Evaluating Counterfactual Explanations:

- Generation Process: Counterfactuals will be generated using the customized DiCE class with PyMoo algorithms, leveraging multi-objective optimization to create explanations that prioritize proximity, sparsity, and diversity. We can use the already implemented functions of DiCE to generate the probability and loss functions.
- Evaluation: The generated counterfactuals will be evaluated on the obejctives we would define in the pymoo class. Comparative analysis with existing DiCE sampling strategies (Genetic Algorithm, Random) needs to be done to compare the performance.

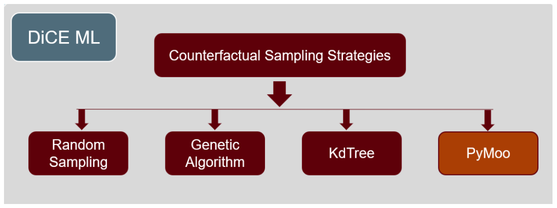


Fig. 1. Integrating Pymoo with DiCEML

B. Modifying the source codes

1) DiCEML: DiCE is a software built on Python that can generate diverse counterfactuals. It is a model agnostic architecture and you can put any models inside to generate your counterfactuals. You need to call the $Dice(data, model, method = random)$ function and pass in your data and model.

In this function, we see how you can call the DiCE functions with a pre-defined model and how you can fix your method to be used for generating counterfactuals. Here, the random method has been chosen. There are other methods like $GeneticAlgorithm$ and $KD-Tree$ that you can choose as your method.

Inside the source code, there are some important functions that we need to understand if we want to integrate a new system.

- *decide_implementation_type* function: Configures which implementation method to use.
- *_generate_counterfactuals* function: Generates the counterfactuals based on the query instances. This is an abstract method that has to be implemented by any new methods.
- *compute_loss* function: Generates the loss function based on *y_loss*, *sparsity_loss* and *proximity_loss*

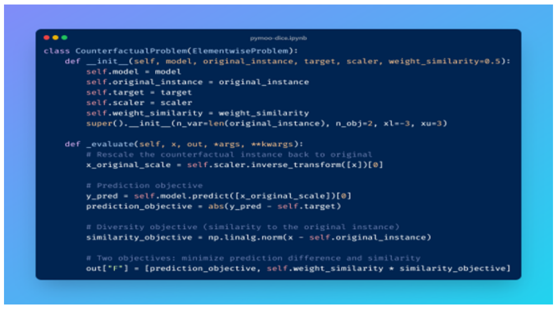
To integrate a new method like Pymoo, we have to modify these functions.

2) Pymoo: Pymoo has multiple classes and functions. But for our project, we have to understand two main classes.

- *problem* class: This class defines the problem and *evaluate* function. You can define your single or multiobjective optimization problem using this class.
- *algorithms* class: This class has all the algorithms that Pymoo has implemented for solving any multiobjective problems. You can use any of these algorithms to solve the problem that you define under the *problemclass*

C. Integrating the Systems

For integrating the systems, we first need to generate counterfactuals using Pymoo. Then we can integrate that function inside DiCEML source code so that we can directly use the Pymoo solution from the DiCE itself. For generating a counterfactual function using Pymoo, we have to define a multiobjective function inside the *_evaluate* function. For simplicity, we have used two objectives for generating counterfactuals: *Minimizing proximity* and *maximizing diversity*. Figure 2 depicts the code snippet of this counterfactual function. We have used $RandomForestRegressor$ as the machine learning model and $CaligorniaHousingDataset$ for this experiment.

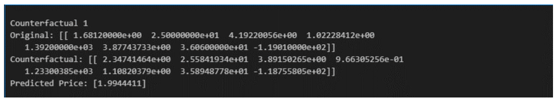


```
class CounterfactualProblem(ElementwiseProblem):
    def __init__(self, model, original_instance, target, scaler, weight_similarity=0.5):
        self.model = model
        self.original_instance = original_instance
        self.target = target
        self.scaler = scaler
        self.weight_similarity = weight_similarity
        super().__init__(n_var=len(original_instance), n_obj=2, xl=-3, xu=3)

    def _evaluate(self, x, out, *args, **kwargs):
        # Rescale the counterfactual instance back to original
        x_original_scale = self.scaler.inverse_transform([x])[0]

        # Prediction objective
        y_pred = self.model.predict([x_original_scale])[0]
        prediction_objective = abs(y_pred - self.target)

        # Diversity objective (similarity to the original instance)
        similarity_objective = np.linalg.norm(x - self.original_instance)

        # Two objectives: minimize prediction difference and similarity
        out["F"] = [prediction_objective, self.weight_similarity * similarity_objective]
```

Fig. 2. Counterfactual generation using Pymoo

Then we have used the $NSGA-II$ algorithm from Pymoo to solve this counterfactual problem and get the counterfactuals. Figure 3 shows the generated counterfactuals from this. It is hard to interpret. So, we plan to see if we can use DiCE's visualization functions with this result.

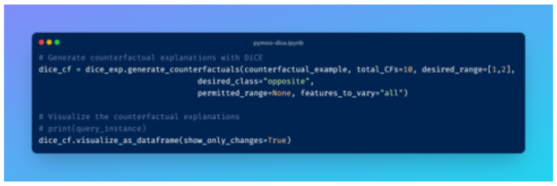
We have used the visualization function from DiCE, figure 4, and put the generated counterfactuals from the Pymoo



```
Counterfactual 1
Original: [[ 1.68120000e+00  2.50000000e+01  4.19220056e+00  1.02228412e+00
   1.39200000e+03  3.87743733e+00  3.60600000e+01 -1.19010000e+02]]
Counterfactual: [[ 2.34741464e+00  2.55841934e+01  3.89150265e+00  9.66305256e-01
   1.23300385e+03  1.10820379e+00  3.58948778e+01 -1.18755805e+02]]
Predicted Price: [1.9944411]
```

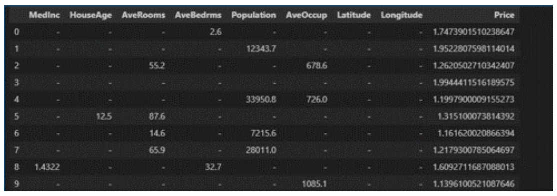
Fig. 3. Generated counterfactuals from Pymoo

model into it. Figure 5 shows the outputs from this procedure. From these results, we can conclude that the outputs are not consistent. Some of the features suggest some unrealistic results, like having 32 bedrooms in a house. This inconsistency could be due to the overly simplistic algorithm we have used to generate the counterfactuals.



```
# Generate counterfactual explanations with DiCE
dice_cf = dice_exp.generate_counterfactuals(counterfactual_example, total_CFs=10, desired_range=[1,2],
                                            desired_class="opposite",
                                            permitted_range=None, features_to_vary="all")

# Visualize the counterfactual explanations
# print(query_instance)
dice_cf.visualize_as_dataframe(show_only_changes=True)
```

Fig. 4. Visualizing the counterfactuals from Pymoo using DiCE

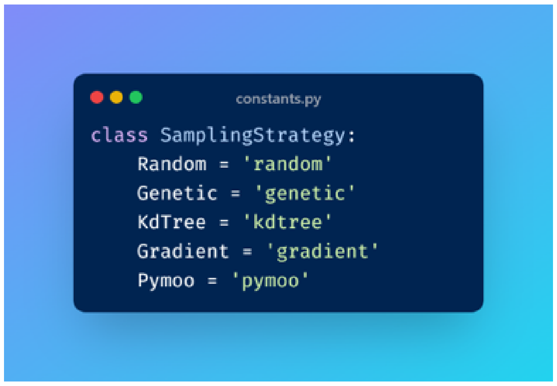


	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Price
0	-	-	-	2.6	-	-	-	-	1.7473901510238647
1	-	-	-	-	12343.7	-	-	-	1.9522807598114014
2	-	-	55.2	-	-	678.6	-	-	1.2620502710342407
3	-	-	-	-	-	-	-	-	1.9944411516189575
4	-	-	-	-	33950.8	726.0	-	-	1.1997900009155273
5	-	12.5	87.6	-	-	-	-	-	1.315100073814392
6	-	-	14.6	-	7215.6	-	-	-	1.161620020866394
7	-	-	65.9	-	28011.0	-	-	-	1.2179300785064697
8	1.4322	-	-	32.7	-	-	-	-	1.6092711687088013
9	-	-	-	-	-	1085.1	-	-	1.1396100521087646

Fig. 5. Visualizing the counterfactuals from Pymoo using DiCE

Till this point, we did not put the Pymoo codes inside the DiCE source code. In the next steps, we gradually changed the source code of DiCE to integrate this Pymoo algorithm for generating counterfactuals. We mainly changed the following 3 functions:

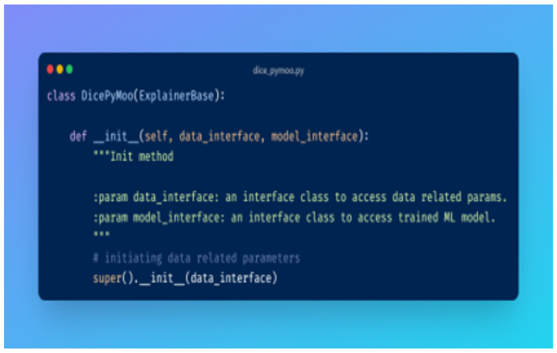
- Defined a new sampling method type as Pymoo in *constants.py* file. The screenshot of a sample code is given at figure 6



```
class SamplingStrategy:
    Random = 'random'
    Genetic = 'genetic'
    KdTree = 'kdtree'
    Gradient = 'gradient'
    Pymoo = 'pymoo'
```

Fig. 6. New sampling method inside DiCE

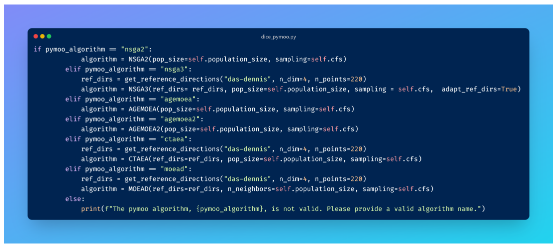
- Created and implemented the $DicePymoo$ class that extends the $ExplainerBase$ class. Started implementing the $\_generate\_counterfactual$ function.



```
class DicePyMoo(ExplainerBase):

    def __init__(self, data_interface, model_interface):
        """Init method

        :param data_interface: an interface class to access data related params.
        :param model_interface: an interface class to access trained ML model.
        """

        # initiating data related parameters
        super().__init__(data_interface)
```

Fig. 7. The Pymoo class to generate the Optimization

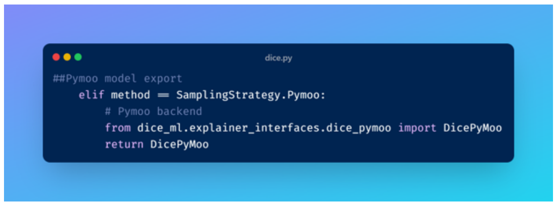
- Built the $Pymoo$ class to generate the optimization function. We have created total 4 objective functions for the optimization. These are: probability, proximity loss, sparsity loss, and diversity loss. Algorithm 1 provides a pseudo code of the optimization function that we have defined.
- Inside the $DicePymoo$ class, we resampled and modified some of the input dimension so that it can fit with the $Pymoo$ algorithm shape. Then, we added another parameter, $pymoo\_algorithm$, in the $\_generate\_counterfactual$ function. This helps us to chose which $Pymoo$ algorithm to use for building the counterfactuals. Figure 8 showcases a snapshot of this.



```
if pymoo_algorithm == "nsga2":
            algorithm = NSGA2(pop_size=self.population_size, sampling=self.cfs)
        elif pymoo_algorithm == "nsga3":
            ref_dirs = get_reference_directions("das-dennis", n_dim=4, n_points=220)
            algorithm = NSGA3(ref_dirs= ref_dirs, pop_size=self.population_size, sampling = self.cfs,  adapt_ref_dirs=True)
        elif pymoo_algorithm == "agemoea":
            algorithm = AGEMOEA(pop_size=self.population_size, sampling=self.cfs)
        elif pymoo_algorithm == "agemoea2":
            algorithm = AGEMOEA2(pop_size=self.population_size, sampling=self.cfs)
        elif pymoo_algorithm == "ctaea":
            ref_dirs = get_reference_directions("das-dennis", n_dim=4, n_points=220)
            algorithm = CTAEA(ref_dirs=ref_dirs, pop_size=self.population_size, sampling=self.cfs)
        elif pymoo_algorithm == "moead":
            ref_dirs = get_reference_directions("das-dennis", n_dim=4, n_points=220)
            algorithm = MOEAD(ref_dirs=ref_dirs, n_neighbors=self.population_size, sampling=self.cfs)
        else:
            print(f"The pymoo algorithm, {pymoo_algorithm}, is not valid. Please provide a valid algorithm name.")
```

Fig. 8. Selecting the Pymoo algorithms

- Added this new method inside the *dice.py* file so that it can be callable from the Dice function. Figure 9 shows a sample screenshot of the calling method.



```
##Pymoo model export
    elif method == SamplingStrategy.Pymoo:
        # Pymoo backend
        from dice_ml.explainer_interfaces.dice_pymoo import DicePyMoo
        return DicePyMoo
```

Fig. 9. Calling the Pymoo method

```
Input  : Data $X$, parameters, and configurations
Output: Updated $out$ with computed objectives and
        losses
/* Step 1: Prediction Objective   */
Compute prediction objectives based on model type
  and desired class.
/* Step 2: Diversity Objective    */
Cluster the data, compute inter-cluster distances, and
  calculate diversity loss.
/* Step 3: Loss Calculation       */
Normalize data, compute total loss based on sparsity
  and proximity metrics.
/* Step 4: Update Output          */
Combine prediction, total loss, and diversity loss into
  the output structure.
```

Algorithm 1: Pseudo code for the evaluation function of the Pymoo optimization functions

D. Tools and Environment Setup

To implement the project, we utilized Python and several Python libraries:

- **Language**: Python was chosen for its extensive support of machine learning and optimization libraries.
- **Libraries**:
  - **DiCE ML**: DiCE (Diverse Counterfactual Explanations) is used to generate counterfactual explanations and provides a flexible framework for integrating custom sampling strategies.
  - **PyMoo**: This library offers a suite of multi-objective optimization algorithms, enhancing the counterfactual sampling process.
  - **Numpy and Pandas**: Used for efficient numerical and data manipulation operations.
  - **sklearn**: Used for calculating different calculations to make the optimization functions work

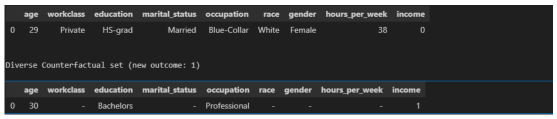
IV. Results

For comparing the results, we have formulated some cases. In the following sections, we will discuss the results and compare them with the regular DiCE generated counterfactuals.

A. Base Case: Generating 1 counterfactual keeping default setting

First of all, we would like to generate the most default setting, with only one counterfactual and keeping all other parameters default. We have used the built in $adults$ dataset for this. For training the model, the $RandomForestClassifier$ model has been used. The data has been preprocessed at first and the continuous and the categorical values have been separated for the models understanding.

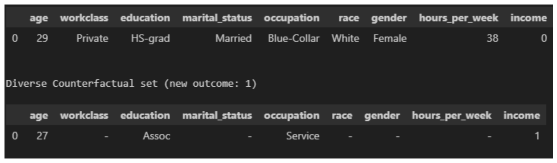
- Original Output: The original output is regular diverse counterfactual output. We have used the $genetic$ method for generating this. Figure 10 demonstrates the output from the original method.



```
   age  workclass  education  marital_status  occupation   race  gender  hours_per_week  income
0   29    Private    HS-grad         Married  Blue-Collar  White  Female             38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass  education  marital_status  occupation    race  gender  hours_per_week  income
0   30          -  Bachelors               -  Professional     -       -               -       1
```

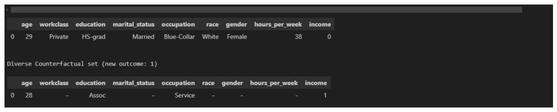
Fig. 10. Generate 1 Counterfactual with Regular GA of DiCE

- Pymoo Output: The Pymoo output is also a very regular diverse counterfactual output. We have used the $NSGA2$ algorithm of Pymoo for generating this. Figure 11 demonstrates the output from the Pymoo method.



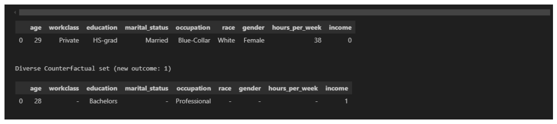
```
   age  workclass  education  marital_status  occupation   race  gender  hours_per_week  income
0   29    Private    HS-grad         Married  Blue-Collar  White  Female             38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass  education  marital_status  occupation  race  gender  hours_per_week  income
0   27          -      Assoc               -     Service     -       -               -       1
```

Fig. 11. Generate 1 Counterfactual with NSGA2 of Pymoo



```
   age  workclass  education  marital_status  occupation   race  gender  hours_per_week  income
0   29    Private    HS-grad         Married  Blue-Collar  White  Female             38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass  education  marital_status  occupation  race  gender  hours_per_week  income
0   28          -      Assoc               -     Service     -       -               -       1
```

Fig. 12. Generate 1 Counterfactual with NSGA3 of Pymoo



```
   age  workclass  education  marital_status  occupation   race  gender  hours_per_week  income
0   29    Private    HS-grad         Married  Blue-Collar  White  Female             38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass  education  marital_status  occupation    race  gender  hours_per_week  income
0   28          -  Bachelors               -  Professional     -       -               -       1
```

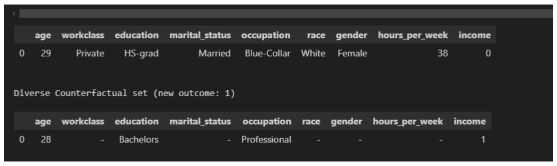
Fig. 13. Generate 1 Counterfactual with AGEMOEA of Pymoo

Figures 12, 13, and 14 demonstrate the outputs from other algorithms of $Pymoo$ also. Overall, for the base case, the primary algorithms of $Pymo$ work really well.

B. Diversity Property: Generating 5 counterfactuals keeping default setting

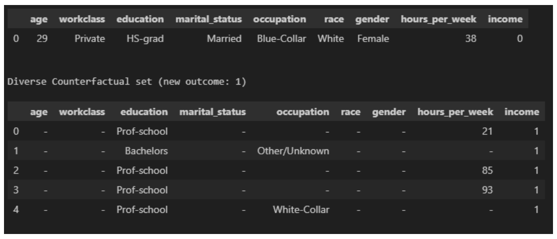
Next, we tested the most diversity setting, by generating 5 counterfactuals while keeping all other parameters default. We have used the same dataset and model here also.

- Original Output: The original output is regular diverse counterfactual output. The outputs are good and diverse. We have used the $random$ method for generating this. Figure 15 demonstrates the output from the original method.
- Pymoo Output: The Pymoo output at first was not very good. We were using the regular diversification functions. But after we used the cluster based diversification, the output improved a lot. We have used the $NSGA2$ algorithm of Pymoo for generating this. Figure 16 demonstrates the output without the diversity loss. And 17 showcases the outputs after the diversity improvement. You can see that the generated counterfactuals are now more diverse.



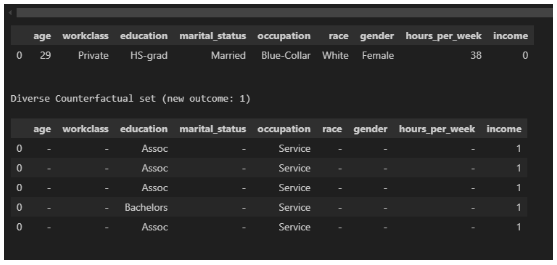
```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   28          -   Bachelors               -  Professional      -       -               -       1
```

Fig. 14. Generate 1 Counterfactual with AGEMOEA2 of Pymoo



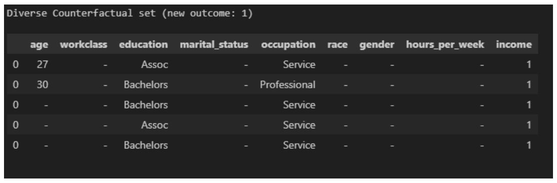
```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0    -          - Prof-school               -             -      -       -              21       1
1    -          -   Bachelors               - Other/Unknown      -       -               -       1
2    -          - Prof-school               -             -      -       -              85       1
3    -          - Prof-school               -             -      -       -              93       1
4    -          - Prof-school               -  White-Collar      -       -               -       1
```

Fig. 15. Generate 5 Counterfactuals with Random method of DiCE



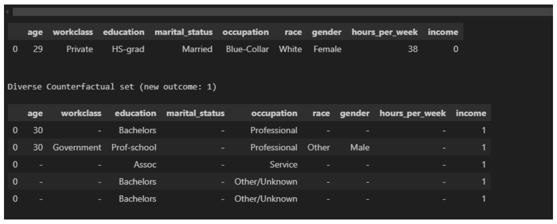
```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -   Bachelors               -       Service      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
```

Fig. 16. Generate 5 Counterfactuals with NSGA2 of Pymoo without the diversity loss



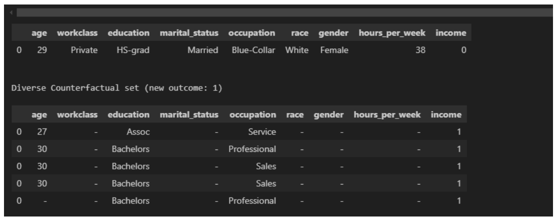
```
Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   27          -       Assoc               -       Service      -       -               -       1
0   30          -   Bachelors               -  Professional      -       -               -       1
0    -          -   Bachelors               -       Service      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -   Bachelors               -       Service      -       -               -       1
```

Fig. 17. Generate 5 Counterfactuals with NSGA2 of Pymoo with the updated diversity loss



```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   30          -   Bachelors               -  Professional      -       -               -       1
0   30 Government Prof-school               -  Professional  Other    Male               -       1
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -   Bachelors               - Other/Unknown      -       -               -       1
0    -          -   Bachelors               - Other/Unknown      -       -               -       1
```

Fig. 18. Generate 5 Counterfactuals with NSGA3 of Pymoo



```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   27          -       Assoc               -       Service      -       -               -       1
0   30          -   Bachelors               -  Professional      -       -               -       1
0   30          -   Bachelors               -         Sales      -       -               -       1
0   30          -   Bachelors               -         Sales      -       -               -       1
0    -          -   Bachelors               -  Professional      -       -               -       1
```

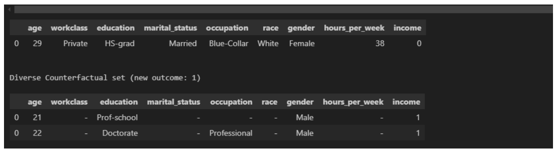
Fig. 19. Generate 5 Counterfactuals with AGEMOEA of Pymoo

Figures 18, and 19 demonstrate the outputs from other algorithms of $Pymoo$ also. Overall, for this case, the primary algorithms of $Pymoo$ work really well after the diversity function improvement .

C. Restraining Features: Fixing which features to vary and fixing the range of desired values

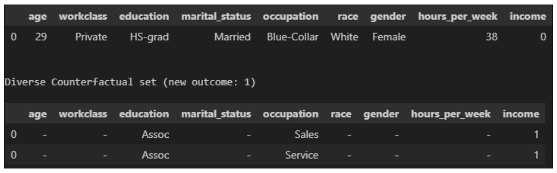
Next, we tested by generating 2 counterfactuals while keeping some features to default and some feature values in a certain range. We have used the same dataset and model here also.

- Original Output: The original output is regular diverse counterfactual output with the ages between 20 to 25 for the figure 20. Figure 21 demonstrates the output where the features education and occupation are allowed to vary. We have used the $genetic$ method for generating this.



```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   21          - Prof-school               -             -      -    Male               -       1
0   22          -   Doctorate               -  Professional      -    Male               -       1
```

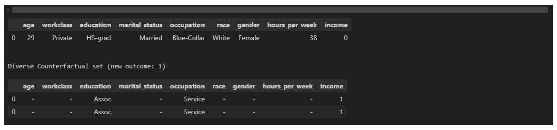
Fig. 20. Generate Counterfactuals with Age and occupation fixed to a range



```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0    -          -       Assoc               -         Sales      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
```

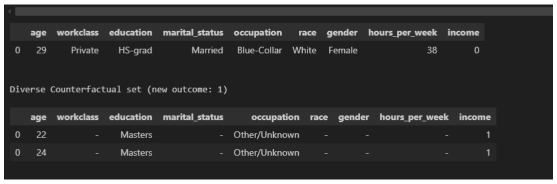
Fig. 21. Generate Counterfactuals with Education and Occupation varying

- Pymoo Output: For Pymoo, we also did the same thing. We varied the features $Education$ and $Occupation$ only. And the age range was given as $20 - 25$ and the occupation range was $Doctorate$ and $Prof - school$.



```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0    -          -       Assoc               -       Service      -       -               -       1
0    -          -       Assoc               -       Service      -       -               -       1
```

Fig. 22. Generate Counterfactuals with NSGA2 of Pymoo with Education and Occupation varying



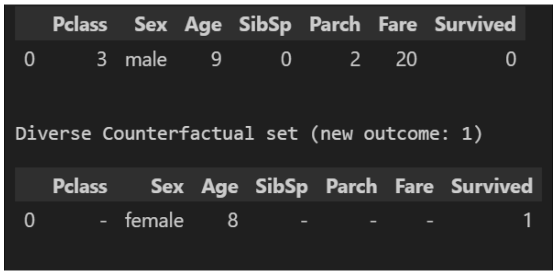
```
   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   29    Private     HS-grad         Married   Blue-Collar  White  Female              38       0


Diverse Counterfactual set (new outcome: 1)

   age  workclass   education  marital_status    occupation   race  gender  hours_per_week  income
0   22          -     Masters               - Other/Unknown      -       -               -       1
0   24          -     Masters               - Other/Unknown      -       -               -       1
```

Fig. 23. Generate Counterfactuals with NSGA2 of Pymoo with Age and occupation fixed to a range

Figures 22, and 23 demonstrate the outputs for this cases. For the $desired_range$ case, the desired range is currently only being achieved for continuous values. For categorical values, it is not giving the exact output that we are expecting.

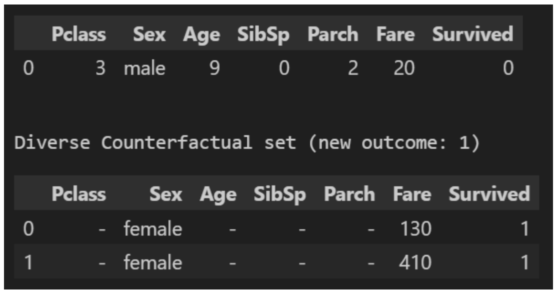
D. Test cases with other dataset

We have also tested with another dataset called the Titanic dataset [6]. The results from this dataset is also promising. We also ran the same 3 scenarios that we have discussed earlier. The sample outputs are given in figures 24, 25 and 26.



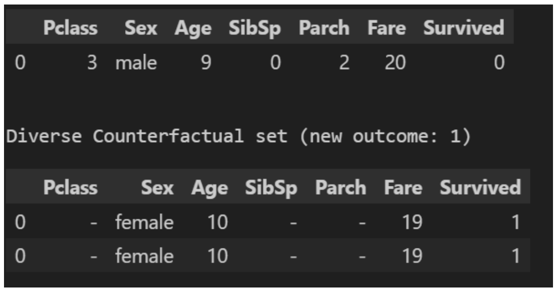
```
     Pclass  Sex   Age  SibSp  Parch  Fare  Survived
0       3   male    9      0      2    20         0


Diverse Counterfactual set (new outcome: 1)

     Pclass    Sex  Age  SibSp  Parch  Fare  Survived
0       -   female    8      -      -     -         1
```

Fig. 24. Generate Counterfactuals with all features in Titanic Data



```
     Pclass  Sex   Age  SibSp  Parch  Fare  Survived
0       3   male    9      0      2    20         0


Diverse Counterfactual set (new outcome: 1)

     Pclass    Sex  Age  SibSp  Parch  Fare  Survived
0       -   female    -      -      -   130         1
1       -   female    -      -      -   410         1
```

Fig. 25. Generate Counterfactuals with some features allowed to vary in Titanic Data



```
     Pclass  Sex   Age  SibSp  Parch  Fare  Survived
0       3   male    9      0      2    20         0


Diverse Counterfactual set (new outcome: 1)

     Pclass    Sex  Age  SibSp  Parch  Fare  Survived
0       -   female   10      -      -    19         1
0       -   female   10      -      -    19         1
```

Fig. 26. Generate Counterfactuals with some features fixed to a range in Titanic Data

Overall, we can say that the integrated $Pymoo$ functions are performing well for the base cases and also when we vary some of the parameters. When we compare it with the $random$ method, the outputs from $Pymoo$ are sometimes even better. When compared with the $genetic$ method, the outputs are quite comparable.

V. Challenges and Risks

There are some challenges we have faced in integrating these two systems. Such as:

- The source code of DiCE is complex. We tested the important and most used features of DiCE. But there are many other features like feature importance, deep learning models etc. that we could not test due to time shortage.
- The *desired_range* feature is working currently only for the continuous features. For categorical features, some other encoding needs to be implemented.
- Some of the algorithm of Pymoo like C-TAEA couldn't be tested as it was taking too much time for each iteration.

VI. Future Works

For our upcoming plans, we plan to finish up the following items:

- We plan to test out other data and models with the Pymoo implementation.
- In future we would like to fix the the *desired_range* feature for categorical features also.
- We plan to test out the remaining pymoo algorithms and if specific modification is needed for them.
- We plan to build a usable library from this which can seamlessly work and generate counterfactuals.

VII. Conclusion

In this study, we successfully integrated PyMOO with DiCEML to improve the generation of counterfactual explanations. PyMOO's algorithms helped us balance multiple objectives such as proximity, sparsity, and diversity. Our results show that this integration improves the quality of counterfactual explanations and makes counterfactuals more meaningful and diverse. Some challenges still remain, such as handling of categorical features and testing additional algorithms and also testing on complex machine learning models like deep neural networks. In the future, we plan to improve our methods and test more datasets to create a more diverse and robust solution. Our work takes a step toward making machine learning models easier to explain and trust.

References

[1] Mothilal, Ramaravind K., Amit Sharma, and Chenhao Tan. "Explaining machine learning classifiers through diverse counterfactual explanations." In Proceedings of the 2020 conference on fairness, accountability, and transparency, pp. 607-617. 2020.

[2] Blank, Julian, and Kalyanmoy Deb. "Pymoo: Multi-objective optimization in python." Ieee access 8 (2020): 89497-89509.

[3] Dandl, Susanne, Christoph Molnar, Martin Binder, and Bernd Bischl. "Multi-objective counterfactual explanations." In International conference on parallel problem solving from nature, pp. 448-469. Cham: Springer International Publishing, 2020.

[4] Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier." In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1135-1144. 2016.

[5] Lundberg, S. M., and S. I. Lee. "A unified approach to interpreting model predictions." NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems. December 2017 [Cited 2021 Jul 20].

[6] https://github.com/datasciencedojo/datasets/blob/master/titanic.csv