The Journal of Open Source Software

CFRL: A Python library for counterfactually fair offline reinforcement learning using data preprocessing

Jianhan Zhang[1], Jitao Wang[2], Chengchun Shi[3], John D. Piette[4], Joshua R. Loftus[3], Donglin Zeng[2], and Zhenke Wu[2¶]

1 Department of Statistics, University of Michigan, USA **2** Department of Biostatistics, University of Michigan, USA **3** Department of Statistics, London School of Economics, UK **4** Department of Health Behavior and Health Equity, School of Public Health, University of Michigan, USA ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Reinforcement learning (RL) aims to learn a sequential decision-making rule, often referred to as a "policy", that maximizes expected discounted cumulative rewards to achieve the highest population-level benefit in an environment across possibly infinitely many time steps. RL has gained popularity for its wide use in fields such as healthcare, banking, autonomous driving, and more recently large language model pre-training. However, the sequential decisions made by an RL algorithm may disadvantage individuals with certain values of a sensitive attribute (e.g., race, ethnicity, gender, education level). The RL algorithm learns an optimal policy that makes decision based on observed state variables. However, if certain value of the sensitive attribute drives the state variables towards values based on which the policy tend to prevent an individual from receiving an action, unfairness will result. For example, Hispanics may under-report pain levels due to cultural factors, misleading the RL agent to assign less therapist time (Piette et al., 2023). More broadly, such concerns about fairness have been raised that the deployment of RL algorithms without careful fairness considerations could erode public trust.

To formally define and address the unfairness problem in sequential decision making settings, Wang et al. (2025) extended the concept of single-stage counterfactual fairness (CF) in a structural causal framework (Kusner et al., 2018) to the multi-stage setting and proposed a data preprocessing algorithm that ensures CF. A policy is CF if, at every time step, the probability of assigning any action does not change had the individual's sensitive attribute take a different value while holding constant other historical exogenous variables and actions. In this light, the data preprocessing algorithm constructs new state variables not impacted by the sensitive attribute(s) to ensure CF. The rewards in data is also preprocessed but not to ensure CF but to improve the value of the learned optimal policy. We refer the readers to Wang et al. (2025) for technical details.

The `CFRL` library implements the data preprocessing algorithm proposed by Wang et al. (2025) and provides a suite of tools to evaluate the value and counterfactual fairness achieved by any given policy. In particular, it reads in data trajectories and outputs preprocessed trajectories, which could then be passed to any off-the-shelf offline RL algorithms to learn an optimal CF policy. The library can also simply read in any policy according to the required format and return its value and level of CF based on the environment (pre-specified or learned from the data).

The Journal of Open Source Software

Statement of Need

Many existing Python libraries implement algorithms that ensure fairness in machine learning. For example, `Fairlearn` (Weerts et al., 2023) and `aif360` (Bellamy et al., 2018) provide tools for mitigating bias in single-stage machine learning predictions under statistical assocoiation-based fairness criterion such as demographic parity and equal opportunity. However, they do not focus on counterfactual fairness, which defines fairness from a causal perspective, and they cannot be easily extended to the reinforcement learning setting in general. Additionally, `ml-fairness-gym` (D'Amour et al., 2020) allows users to simulate unfairness in sequential decision-making, but it neither implements algorithms that reduce unfairness nor addresses counterfactual fairness. To our current knowledge, Wang et al. (2025) is the first work to study counterfactual fairness in reinforcement learning. Correspondingly, `CFRL` is also the first code library to address counterfactual fairness in the reinforcement learning setting.

The contribution of CFRL is two-fold. First, it implements a data preprocessing algorithm that removes bias from offline RL training data. For each individual (or sample) in the data, the preprocessing algorithm estimates the counterfactual states under different sensitive attribute values and concatenates all of the individual's counterfactual states into a new state variable. The preprocessed data can then be directly used by existing RL algorithms for policy learning, and the learned policy should be approximately counterfactually fair. Second, it provides a platform for assessing RL policies based on counterfactual fairness. After passing in a policy and a trajectory dataset from the environment of interest, users can estimate the discounted cumulative reward and level of counterfactual fairness achieved by the policy in the environment of interest. This not only allows stakeholders to test their fair RL policies before deployment but also offers RL researchers a hands-on tool to evaluate newly developed counterfactually fair RL algorithms.

High-level Design

The `CFRL` library is composed of 5 major modules. The functionalities of the modules are summarized in the table below.

Module	Functionalities

Module	Functionalities
`reader`	Implements functions that read tabular trajectory data from either a `.csv` file or a `pandas.Dataframe` into an array format required by `CFRL`. Also implements functions that export trajectory data to either a `.csv` file or a `pandas.Dataframe`.
`preprocessor`	Implements the data preprocessing algorithm introduced in Wang et al. (2025).
`agents`	Implements a fitted Q-iteration (FQI) algorithm, which learns RL policies and makes decisions based on the learned policy. Users can also pass a preprocessor to the FQI; in this case, the FQI will be able to take in unpreprocessed trajectories, internally preprocess the input trajectories, and directly output counterfactually fair policies.
`environment`	Implements a synthetic environment that produces synthetic data as well as a simulated environment that estimates and simulates the transition dynamics of the unknown environment underlying some real-world RL trajectory data. Also implements real-world RL trajectory data. Also implements functions for sampling trajectories from the synthetic and simulated environments.
`fqe`	Implements a fitted Q-evaluation (FQE) algorithm, which can be used to evaluate the value of a policy.
`evaluation`	Implements functions that evaluate the value and counterfactual fairness of a policy. Depending on the user's needs, the evaluation can be done either in a synthetic environment or in a simulated environment.

A general CFRL workflow is as follows: First, simulate a trajectory using `environment` or read in a trajectory using `reader`. Then, train a preprocessor using `preprocessor` to remove the bias in the trajectory data. After that, pass the preprocessed trajectory into the FQI algorithm in `agents` to learn a counterfactually fair policy. Finally, use functions in `evaluation` to evaluate the value and counterfactual fairness of the trained policy.

Data Example

We provide a data example to demontrate how `CFRL` learns a counterfactually fair policy from real-world trajectory data with unknown underlying transition dynamics. We also show how `CFRL` evaluates the value and counterfactual fairness of the learned policy. We note that this is only one of the many workflows that `CFRL` can perform. For example, `CFRL` can also generate synthetic trajectory data and use it to evaluate the value and counterfactual fairness resulting from some custom data preprocessing methods. We refer interested readers to the "Example Workflows" section of the CFRL documentation for more workflow examples.

Data Loading

In this demonstration, we use an offline trajectory generated from a `SyntheticEnvironment` following some pre-specified transition rules. Although it is actually synthesized, we treat it as if it is from some unknown environment for pedagogical convenience.

The trajectory contains 500 individuals (i.e. $N = 500$) and 10 transitions (i.e. $T = 10$). The sensitive attribute variable and the state variable are both univariate. The sensitive attributes are binary (0 or 1). The actions are also binary (0 or 1) and were sampled using a policy that selects 0 or 1 randomly with equal probability. The trajectory is stored in a tabular format in a `.csv` file. We use `read_trajectory_from_csv()` to load the trajectory from the `.csv` format into the array format required by `CFRL`.

```
zs, states, actions, rewards, ids = read_trajectory_from_csv(
    path='../data/sample_data_large_uni.csv', z_labels=['z1'],
```

```
    state_labels=['state1'], action_label='action', reward_label='reward',
    id_label='ID', T=10)
```

We then split the trajectory data into a training set (80%) and a testing set (20%) using scikit-learn's `train_test_split()`. The training set is used to train the counterfactually fair policy, while the testing set is used to evaluate the value and counterfactual fairness achieved by the policy.

```
(zs_train, zs_test, states_train, states_test,
 actions_train, actions_test, rewards_train, rewards_test
) = train_test_split(zs, states, actions, rewards, test_size=0.2)
```

Preprocessor Training & Trajectory Preprocessing

We now train a `SequentialPreprocessor` and preprocess the trajectory. The `SequentialPreprocessor` ensures the learned policy is counterfactually fair by removing the bias from the training trajectory data. Due to limited trajectory data, the data to be processed will also be the data used to train the preprocessor, so we set `cross_folds=5` to reduce overfitting. In this case, `train_preprocessor()` will internally divide the training data into 5 folds, and each fold is preprocessed using a model that is trained on the other 4 folds. We initialize the `SequentialPreprocessor`, and `train_preprocessor()` will take care of both preprocessor training and trajectory preprocessing.

```
sp = SequentialPreprocessor(z_space=[[0], [1]], num_actions=2, cross_folds=5,
                            mode='single', reg_model='nn')
states_tilde, rewards_tilde = sp.train_preprocessor(
    zs=zs_train, xs=states_train, actions=actions_train, rewards=rewards_train)
```

As an aside, we remark that in the case where the trajectories to be preprocessed are separate from the trajectories used to train the preprocessor, we should typically set `cross_folds=1`. Then we use `train_preprocessor()` to train the preprocessor and use `preprocess_multiple_steps()` to preprocess the trajectories.

Counterfactually Fair Policy Learning

Now we train a counterfactually fair policy using the preprocessed data and `FQI` with `sp` as its internal preprocessor. By default, the input data will first be preprocessed by `sp` before being used for policy learning. In our case, since the training data `state_tilde` and `rewards_tilde` are already preprocessed, we set `preprocess=False` during training so that the input trajectory will not be preprocessed again by the internal preprocessor (i.e. `sp`).

```
agent = FQI(num_actions=2, model_type='nn', preprocessor=sp)
agent.train(zs=zs_train, xs=states_tilde, actions=actions_train,
            rewards=rewards_tilde, max_iter=100, preprocess=False)
```

`SimulatedEnvironment` Training

Before moving on to the evaluation stage, there is one more thing to do: We need to train a `SimulatedEnvironment` that mimics the transition rules of the true environment that generated the training trajectory, which will be used by the evaluation functions to simulate the true data-generating environment. To do so, we initialize a `SimulatedEnvironment` and train it on the whole trajectory data (i.e. training set and testing set combined).

```
env = SimulatedEnvironment(num_actions=2, state_model_type='nn',
                           reward_model_type='nn')
env.fit(zs=zs, states=states, actions=actions, rewards=rewards)
```

Value and Counterfactual Fairness Evaluation

We now use `evaluate_value_through_fqe()` and `evaluate_fairness_through_model()` to estimate the value and counterfactual fairness achieved by the trained policy when interacting with the environment of interest, respectively. The counterfactual fairness is represented by a metric from 0 to 1, with 0 representing perfect fairness and 1 indicating complete unfairness. We use the testing set for evaluation.

```
value = evaluate_reward_through_fqe(zs=zs_test, states=states_test,
    actions=actions_test, rewards=rewards_test, policy=agent, model_type='nn')
cf_metric = evaluate_fairness_through_model(env=env, zs=zs_test, states=states_test,
                                            actions=actions_test, policy=agent)
```

The estimated value is 7.358 and CF metric is 0.042, which indicates our policy is close to being perfectly counterfactually fair. Indeed, the CF metric should be exactly 0 if we know the true dynamics of the environment of interest; the reason why it is not exactly 0 here is because we need to estimate the dynamics of the environment of interest during preprocessing, which can introduce errors.

Comparisons Against Baseline Methods

We can compare the sequential data preprocessing method in `CFRL` against a few baselines: Random, which selects each action randomly with equal probability; Full, which uses all variables, including the sensitive attribute, for policy learning; and Unaware, which uses all variables except the sensitive attribute for policy learning. We implemented these baselines and evaluated their values and CF metrics as part of the code example of the "Assessing Policies Using Real Data" workflow in the "Example Workflows" section of the CFRL documentation. We summarize below the values and CF metrics calculated in this code example.

	Random	Full	Unaware
Value	-1.444	8.606	8.588
CF Metric	0	0.407	0.446

For any individual, we assume all of his or her counterfactual trajectories share the same randomness. Thus, the "random" baseline always selects the same action in all the counterfactual trajectories for the same individual, resulting in perfect fairness. On the other hand, the other two baselines both led to much less fair policies than our preprocessing method, which suggests that our preprocessing method likely reduced the bias in the training trajectory effectively.

Conclusions

`CFRL` is a Python library that empowers counterfactually fair reinforcement learning through data preprocessing. It also provides tools to evaluate the value and counterfactual fairness of a given policy. As far as we know, it is the first library to address counterfactual fairness problems in the context of reinforcement learning. Nevertheless, despite this, `CFRL` also admits a few limitations. For example, the current `CFRL` implementation requires every individual in the offline dataset to have the same number of time steps. Extending the library to accommodate variable-length episodes can improve its flexibility and usefulness. Besides, `CFRL` could also be made more well-rounded by integrating the preprocessor with popular offline RL algorithm libraries such as `d3rlpy` (Seno & Imai, 2022), or connecting the evaluation functions with established RL environment libraries such as `gym` (Towers et al., 2024). We leave these extensions to future updates.

References

Bellamy, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K. N., Richards, J., Saha, D., Sattigeri, P., Singh, M., Varshney, K. R., & Zhang, Y. (2018). *AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias*. https://arxiv.org/abs/1810.01943

D'Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., & Halpern, Y. (2020). Fairness is not static: Deeper understanding of long term fairness via simulation studies. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 525–534. https://doi.org/10.1145/3351095.3372878

Kusner, M. J., Loftus, J. R., Russell, C., & Silva, R. (2018). *Counterfactual fairness*. https://arxiv.org/abs/1703.06856

Piette, J. D., Thomas, L., Newman, S., Marinec, N., Krauss, J., Chen, J., Wu, Z., & Bohnert, A. S. B. (2023). An automatically adaptive digital health intervention to decrease opioid-related risk while conserving counselor time: Quantitative analysis of treatment decisions based on artificial intelligence and patient-reported risk measures. *J Med Internet Res*, *25*, e44165. https://doi.org/10.2196/44165

Seno, T., & Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, *23*(315), 1–20. http://jmlr.org/papers/v23/22-0017.html

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., & others. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv Preprint arXiv:2407.17032*.

Wang, J., Shi, C., Piette, J. D., Loftus, J. R., Zeng, D., & Wu, Z. (2025). *Counterfactually fair reinforcement learning via sequential data preprocessing*. https://arxiv.org/abs/2501.06366

Weerts, H., Dudík, M., Edgar, R., Jalali, A., Lutz, R., & Madaio, M. (2023). Fairlearn: Assessing and improving fairness of AI systems. In *Journal of Machine Learning Research* (No. 257; Vol. 24, pp. 1–8). http://jmlr.org/papers/v24/23-0389.html