

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

Jianhan Zhang¹, Jitao Wang², Chengchun Shi³, John D. Piette⁴, Joshua R. Loftus³, Donglin Zeng², 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](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Open Journals](#) ↗

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

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). This is because the RL algorithm learns an optimal policy that makes decisions based on observed state variables. If certain value of the sensitive attribute influences the state variables in a way that leads the policy to systematically withhold an action from an individual, unfairness will result. For example, Hispanics may under-report their pain levels due to cultural factors, misleading the RL agent to assign less therapist time to them (Piette et al., 2023). Deployment of RL algorithms without careful fairness considerations can raise concerns and 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 taken a different value, while holding constant other historical exogenous variables and actions. In this light, the data preprocessing algorithm ensures CF by constructing new state variables that are not impacted by the sensitive attribute(s). The rewards in data are also preprocessed, but the purpose of preprocessing the rewards is to improve the value of the learned optimal policy rather than ensure CF. We refer interested readers to Wang et al. (2025) for more 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 CF level achieved by any given policy. The library produces preprocessed trajectories that can be used by any off-the-shelf offline RL algorithms, such as fitted Q-iteration (FQI) (Riedmiller, 2005), to learn an optimal CF policy. The library can also simply read in any policy following a required format and return its value and CF level in the environment of interest, where the environment can be either pre-specified or learned from the data.

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 association-based fairness criterion such as demographic parity and equal opportunity. However, they do not focus on counterfactual fairness, which defines an individual-level fairness concept 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 CF. To our knowledge, Wang et al. (2025) is the first work to study CF in RL. Correspondingly, CFRL is also the first code library to address CF in the RL setting.

The contribution of CFRL is two-fold. First, it implements a data preprocessing algorithm that ensures CF in offline RL. For each individual in the data, the preprocessing algorithm sequentially estimates the counterfactual states under different sensitive attribute values and concatenates all of the individual's counterfactual states into a new state vector. The preprocessed data can then be directly used by existing RL algorithms for policy learning, and the learned policy should be counterfactually fair up to finite-sample estimation accuracy. Second, it provides a platform for assessing RL policies based on CF. After passing in any policy and a data trajectory from the environment of interest, users can estimate the value and CF level achieved by the policy in the environment of interest.

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
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 (Riedmiller, 2005), 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 functions for sampling trajectories from the synthetic and simulated environments.
evaluation	Implements functions that evaluate the value and CF level 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 and preprocess the training 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 CF level of the trained policy.

Data Example

We provide a data example to demonstrate 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 CF level 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 CF level resulting from some custom data preprocessing methods. We refer interested readers to the “[Example Workflows](#)” section of the CFRL documentation for more workflow examples.

Load Data

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 attribute is binary (0 or 1). The actions are also binary (0 or 1) and were sampled using a behavior 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 CF level 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)
```

Train Preprocessor & Preprocess Trajectories

We now train a `SequentialPreprocessor` and preprocess the trajectory. The `SequentialPreprocessor` ensures the learned policy is counterfactually fair by constructing new state variables that are not impacted by the sensitive attribute. Due to limited trajectory data, the data to be preprocessed 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)
```

Counterfactually Fair Policy Learning

100 Now we train a counterfactually fair policy using the preprocessed data and FQI with `sp` as its
 101 internal preprocessor. By default, the input data will first be preprocessed by `sp` before being
 102 used for policy learning. However, since the training data `state_tilde` and `rewards_tilde`
 103 are already preprocessed in our case, we set `preprocess=False` during training so that the
 104 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)
```

105 SimulatedEnvironment Training

106 Before moving on to the evaluation stage, there is one more thing to do: We need to train a
 107 `SimulatedEnvironment` that mimics the transition rules of the true environment that generated
 108 the training trajectory, which will be used by the evaluation functions to simulate the true
 109 data-generating environment. To do so, we initialize a `SimulatedEnvironment` and train it on
 110 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)
```

111 Value and Counterfactual Fairness Evaluation

112 We now use `evaluate_value_through_fqe()` and `evaluate_fairness_through_model()` to
 113 estimate the value and CF level achieved by the trained policy when interacting with the
 114 environment of interest, respectively. The CF level is represented by a metric from 0 to 1, with
 115 0 representing perfect fairness and 1 indicating complete unfairness. We use the testing set for
 116 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)
```

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

122 Comparisons Against Baseline Methods

123 We can compare the sequential data preprocessing method in CFRL against a few baselines:
 124 Random, which selects each action randomly with equal probability; Full, which uses all
 125 variables, including the sensitive attribute, for policy learning; and Unaware, which uses all
 126 variables except the sensitive attribute for policy learning. We implemented these baselines
 127 and evaluated their values and CF levels as part of the code example of the “Assessing Policies
 128 Using Real Data” workflow in the “[Example Workflows](#)” section of the CFRL documentation.
 129 We summarize below the values and CF metrics calculated in this code example, where “ours”
 130 stands for the `SequentialPreprocessor`.

	Random	Full	Unaware	Ours
Value	-1.444	8.606	8.588	7.358
CF Metric	0	0.407	0.446	0.042

By definition, the “random” baseline always achieves perfect CF. On the other hand, “ours” resulted in much fairer policies than “full” and “unaware”, which suggests that the SequentialPreprocessor can effectively improve CF. Nevertheless, as a trade-off for higher CF, “ours” achieved a lower value than “full” and “unaware”.

Conclusions

CFRL is a Python library that enables counterfactually fair reinforcement learning through data preprocessing. It also provides tools to evaluate the value and CF level of a given policy. To our knowledge, it is the first library to address CF problems in the context of RL. Nevertheless, despite this, CFRL also has 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 by 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>
- Riedmiller, M. (2005). Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, & L. Torgo (Eds.), *Machine learning: ECML 2005* (pp. 317–328). Springer Berlin Heidelberg. ISBN: 978-3-540-31692-3
- 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*

DRAFT