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

Jianhan Zhang[1], **Jitao Wang**[2], **Chengchun Shi**[3], **John D. Piette**[4], **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

Summary

Reinforcement learning (RL) aims to learn and evaluate a sequential decision-making rule, often referred to as a "policy", that maximizes expected discounted cumulative rewards to optimize population-level benefit in an environment across possibly infinitely many time steps. RL has gained popularity 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, income, gender, education level). An RL algorithm learns an optimal policy that makes decisions based on observed state variables. If certain values of the sensitive attribute influence the state variables in a way that influences the policy, such as by leading the policy to systematically withhold services from an individual, unfairness will result. For example, Hispanics may under-report their pain levels due to cultural factors, misleading a fairness-blind 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 in high-stake settings.

To formally define and address the fairness 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 counterfactually fair 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 reward values in the 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 counterfactual unfairness level achieved by any given policy. Here, "CFRL" stands for "Counterfactual Fairness in Reinforcement Learning". The library produces preprocessed trajectories that can be used by an off-the-shelf offline RL algorithm, 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 (i.e. expected discounted cumulative reward) and counterfactual unfairness level in the environment of interest, where the environment can be either pre-specified or learned from

44 the data.

Statement of Need

46 Many existing Python libraries implement algorithms designed to ensure fairness in machine
47 learning. For example, Fairlearn (Weerts et al., 2023) and aif360 (Bellamy et al., 2018)
48 provide tools for mitigating bias in single-stage machine learning predictions under statistical
49 association-based fairness criteria such as demographic parity and equal opportunity. However,
50 existing libraries do not focus on counterfactual fairness, which defines an individual-level
51 fairness concept from a causal perspective, and they cannot be easily extended to the generalized
52 reinforcement learning setting. Scripts available from ml-fairness-gym (D'Amour et al., 2020)
53 allow users to simulate unfairness in sequential decision-making, but they neither implement
54 algorithms that reduce unfairness nor address CF. To our knowledge, Wang et al. (2025) is
55 the first work to study CF in RL. Correspondingly, CFRL is also the first code library to address
56 CF in the RL setting.

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

High-level Design

67 The CFRL library is composed of 5 major modules. The functionalities of the modules are
68 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 counterfactual unfairness 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.

69 A general CFRL workflow is as follows: First, simulate a trajectory using environment or read in

70 a trajectory using `reader`. Then, train a preprocessor using `preprocessor` and preprocess the
71 training trajectory data. After that, pass the preprocessed trajectory into the FQI algorithm in
72 `agents` to learn a counterfactually fair policy. Finally, use functions in `evaluation` to evaluate
73 the value and counterfactual unfairness level of the trained policy.

74 In addition, CFRL also provides tools to check for potential non-convergence that arise in
75 neural network training, FQI, and FQE. More discussions about the sources, checks, and
76 fixes of non-convergence in CFRL can be found in the "Common Issues" section of the CFRL
77 documentation.

Data Example

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

87 We also recorded the computing times of different workflows under different combinations of
88 the number of individuals (N) and the number of transitions (T) in the "Computing Times"
89 section of the CFRL documentation. For example, under $N = 500$ and $T = 10$, the workflow
90 presented in this data example (which is also the "real data workflow" in the documentation)
91 ran for 378.62 seconds on average in our computing environment.

92 Load Data

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

96 The trajectory contains 500 individuals (i.e. $N = 500$) and 10 transitions (i.e. $T = 10$). The
97 sensitive attribute variable and the state variable are both univariate. The sensitive attribute
98 is binary (0 or 1). The actions are also binary (0 or 1) and were sampled using a behavior
99 policy that selects 0 or 1 randomly with equal probability. The trajectory is stored in a tabular
100 format in a `.csv` file. We use `read_trajectory_from_csv()` to load the trajectory from the
101 `.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)
```

102 We then split the trajectory data into a training set (80%) and a testing set (20%) using
103 scikit-learn's `train_test_split()`. The training set is used to train the counterfactually fair
104 policy, while the testing set is used to evaluate the value and counterfactual unfairness level
105 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)
```

106 Train Preprocessor & Preprocess Trajectories

107 We now train a `SequentialPreprocessor` and preprocess the trajectory. The `SequentialPreprocessor`
108 ensures the learned policy is counterfactually fair by constructing new state variables that

¹⁰⁹ are not impacted by the sensitive attribute. Due to limited trajectory data, the data to be
¹¹⁰ preprocessed will also be the data used to train the preprocessor, so we set `cross_folds=5`
¹¹¹ to reduce overfitting. In this case, `train_preprocessor()` will internally divide the training
¹¹² data into 5 folds, and each fold is preprocessed using a model that is trained on the other 4
¹¹³ folds. We initialize the `SequentialPreprocessor`, and `train_preprocessor()` will take care
¹¹⁴ of both preprocessor training and trajectory preprocessing.

```python
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)
```

¹¹⁵ Counterfactually Fair Policy Learning

¹¹⁶ Next we train a counterfactually fair policy using the preprocessed data and `FQI` with `sp` as its
¹¹⁷ internal preprocessor. By default, the input data will first be preprocessed by `sp` before being
¹¹⁸ used for policy learning. However, since the training data `state_tilde` and `rewards_tilde`
¹¹⁹ are already preprocessed in our case, we set `preprocess=False` during training so that the
¹²⁰ input trajectory will not be preprocessed again by the internal preprocessor (i.e. `sp`).

```python
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)
```

¹²¹ `SimulatedEnvironment` Training

¹²² Before moving on to the evaluation stage, there is one more step: We need to train a
¹²³ `SimulatedEnvironment` that mimics the transition rules of the true environment that generated
¹²⁴ the training trajectory, which will be used by the evaluation functions to simulate the true
¹²⁵ data-generating environment. To do so, we initialize a `SimulatedEnvironment` and train it on
¹²⁶ the whole trajectory data (i.e. training set and testing set combined).

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

¹²⁷ Value and Counterfactual Unfairness Level Evaluation

¹²⁸ We now use `evaluate_value_through_fqe()` and `evaluate_fairness_through_model()` to
¹²⁹ estimate the value and counterfactual unfairness level achieved by the trained policy when
¹³⁰ interacting with the environment of interest, respectively. The counterfactual unfairness level
¹³¹ is represented by a metric from 0 to 1, with 0 representing perfect fairness and 1 indicating
¹³² complete unfairness. We use the testing set for evaluation.

```python
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)
```

¹³³ The estimated value is 7.358 and CF metric is 0.042, which indicates our policy is close to
¹³⁴ being perfectly counterfactually fair. Indeed, the CF metric should be exactly 0 if we know
¹³⁵ the true dynamics of the environment of interest; the reason why it is not exactly 0 here is
¹³⁶ because we need to estimate the dynamics of the environment of interest during preprocessing,
¹³⁷ which can introduce finite-sample errors.

¹³⁸ Comparisons Against Baseline Methods

¹³⁹ We can compare the sequential data preprocessing method in CFRL against a few baselines:
¹⁴⁰ Random, which selects each action randomly with equal probability; Full, which uses all

141 variables, including the sensitive attribute, for policy learning; and Unaware, which uses all
142 variables except the sensitive attribute for policy learning. We implemented these baselines
143 and evaluated their values and counterfactual unfairness levels as part of the code example of
144 the "Assessing Policies Using Real Data" workflow in the "Example Workflows" section of the
145 CFRL documentation. We summarize below the values and CF metrics calculated in this code
146 example, where "ours" stands for the SequentialPreprocessor.

	Random	Full	Unaware	Ours
Value	−1.444	8.606	8.588	7.358
Counterfactual Unfairness Level	0	0.407	0.446	0.042

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

Conclusions

152 CFRL is a Python library that enables CF reinforcement learning through data preprocessing.
153 It also provides tools to calculate the value and unfairness level of a given policy. To our
154 knowledge, it is the first library to address CF problems in the context of RL. The practical
155 utility of CFRL can be further improved via extensions. First, the current CFRL implementation
156 requires every individual in the offline dataset to have the same number of time steps. Extending
157 the library to accommodate variable-length episodes can improve its flexibility and usefulness.
158 Second, CFRL can further combine the preprocessor with popular offline RL algorithm libraries
159 such as d3rlpy (Seno & Imai, 2022), or connect the evaluation functions with established RL
160 environment libraries such as gym (Towers et al., 2024). Third, generalization to non-additive
161 counterfactual states reconstruction can make CFRL theoretically more versatile. We leave
162 these extensions to future updates.

References

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

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

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

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

179 Riedmiller, M. (2005). Neural fitted Q iteration – first experiences with a data efficient neural
180 reinforcement learning method. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, & L.

181   Torgo (Eds.), *Machine learning: ECML 2005* (pp. 317–328). Springer Berlin Heidelberg.
182   ISBN: 978-3-540-31692-3

183 Seno, T., & Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library. *Journal of*
184   *Machine Learning Research*, *23*(315), 1–20.

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

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

190 Weerts, H., Dudík, M., Edgar, R., Jalali, A., Lutz, R., & Madaio, M. (2023). Fairlearn:
191   Assessing and improving fairness of AI systems. In *Journal of Machine Learning Research*
192   (No. 257; Vol. 24, pp. 1–8).