Assessing the Risk of Discriminatory Bias in Classification Datasets – Supplementary Materials –

Kejun Dai, Jonathan Kim, Sašo Džeroski, Jörg Wicker, Gillian Dobbie, and Katharina Dost

1 Dataset Generation

We opt to generate synthetic datasets to support our training of meta-learning models [1]. In order to generate data points that are dependent on some features and independent of others, we first construct a hypercube whose dimensions include dependent features. Then, we create clusters of normally distributed data points around the vertices of the hypercube. At last, we assign equal numbers of clusters to each class of label values. We also parameterize our dataset generation process to promote more diversity among our synthesized datasets. For example, we can reduce the number of dependent features and increase the number of independent features to generate a richer classification dataset.

In practice, the parameters to generate synthetic datasets will be randomly drawn from two parameter pools; one represents datasets with small size and small complexity, and the other represents datasets with larger size and larger complexity. The parameter pools for both types of synthetic datasets are detailed in Table 1. After generation, we will then randomly mark at most half of the possible label values as positive labels.

2 Bias Generation

When generating synthetic bias for the synthetic datasets, we alternate between the following two methods. *Patch sensitive complex generation*'s intuition is to divide the datasets into many small chunks and then assemble them into the desired output. The algorithm first partitions the datasets into many small equal-size sensitive shapes. For each requested sensitive complex, the algorithm will use greedy search to find the best-performing shape from the remaining unused shape without overshooting the required size and prevalence and add it to the complex. The end output of the algorithm will be disjointedly connected to sensitive complexes. The results of this algorithm also do not overlap with each other. Two examples are provided in Figure 1.

The goal of *continuous sensitive complex generation* is to output a singular sensitive shape as the sensitive complex that closely matches the size and prevalence requirement. For each requested sensitive complex, the algorithm will choose a set of random features and iterate through them. For each iterated feature, the algorithm will find the best five suitable sensitive rules that are close to the requirements but

[1] For our experiments, we use scikit learn's *make_classification* method to generate biased datasets.

	Small dataset	**Large dataset**
# of data points	2000-10000	10000-20000
# of classes	2-6	2-10
# of features	2-10	10-50
# of dependent features	≥ 1 $<$ #features	$\geq \frac{1}{10}$#features $<$ #features
# of independent features	≥ 1 $<$ #features	≥ 1 $<$ #features
# of repeated features	0	$\leq \frac{1}{5}$#features
# of clusters per class	1-3	1-3

Table 1: Parameter pools for our synthetic dataset generation process. All parameters will be validated and adjusted before running the method.

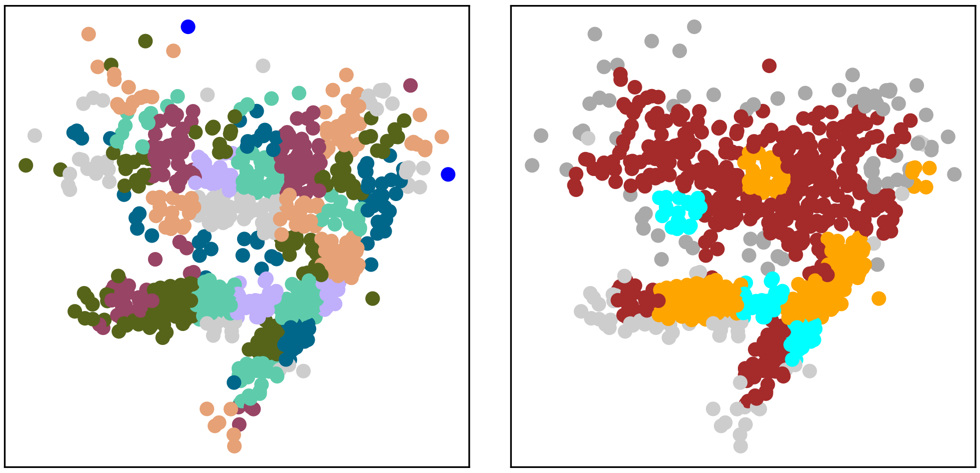

Fig. 1: Example for *Patch sensitive complex generation*

also overshooting to allow further dissection by other sensitive rules. After all iterations of features, the algorithm will combine them into a sensitive shape as output. The end output of the algorithm will be a continuous space that closely matches the specified size and prevalence. However, they can overlap with each other and can have a consequence that is unseen in sensitive attribute frameworks. See Figure 2 for examples.

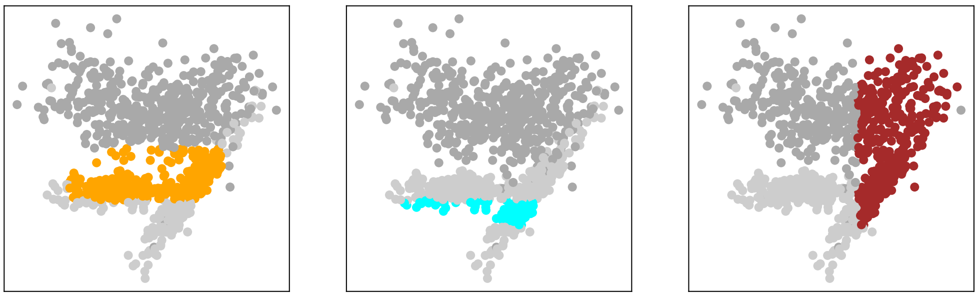
Fig. 2: Example for *continuous sensitive complex generation*

3 Hyperparameter Tuning

In our experiment, we use grid search[2] to tune hyperparameters of Random Forest, XGBoost, LightGBM, and Multi-Layered Perceptron (MLP) models. For each model we train, we first designate the models' baseline hyperparameters, which set up their training procedure, such as *random_state* or *device_type*. Hyperparameters that influence the models' behavior and thus performance will then be selected from ranges of possible candidates through grid search. The search grids for each model are detailed in Table 2.

4 Other Error Metrics for Regression Tasks

In addition to *Root Mean Squared Error (RMSE)*, we also use error metrics of *Mean Absolute Error (MAE)*, *Mean Squared Error (MSE)*, *Mean Absolute Percentage Error (MAPE)*, *Symmetric Mean Absolute Percentage Error (SMAPE)*, *Mean Squared Logistic Error (MSLE)* to evaluate the performance of our meta-learning models in our experiments. However, they show similar results to RMSE and reach the same conclusions. Note that since bias metrics are range from 0 to 2, the MAPE of our meta-learning model sometimes explodes exponentially and is impossible to comprehend. These error metrics' results are detailed in Figure 3.

5 Legend of Meta-features

When constructing our meta-dataset, we use the Python library *pymfe*[3] to extract meta-features. The meaning of the meta-features is as follows[4]:

1. Complexity meta-features
   - **c1**: The entropy of class proportions.
   - **c2**: The imbalance ratio .
   - **cls_coef**: Clustering coefficient.
   - **density**: Average density of the network for synthesised datasets.
   - **f1.mean**: Average maximum Fisher's discriminant ratio .

[2]We use scikit learn's *GridSearchCV* implementation to tune models' hyperparameters
[3]github.com/ealcobaca/pymfe
[4]Further details on the meta-features can be found at pymfe.readthedocs.io/en/latest/auto_pages/meta_features_description.html

Random Forest	n_estimators: [25, 50, 75, 100, 125] max_depth: [5, 10, 20, 30, 40] max_features:[#features, $\frac{\#\text{feature}}{2}$, $\sqrt{\#\text{feature}}$]
XGBoost	n_estimators: [50, 100, 150, 200, 250] max_depth: [2, 5, 10, 15, 20] learning_rate: [0.01, 0.3, 0.5]
LightGBM	n_estimators: [50, 100, 150, 200, 250, 300, 500, 750, 1000] max_depth: [2, 3, 5, 7, 10, 15, 20, 30] learning_rate: [0.01, 0.1, 0.5]
MLP	hidden_layer_sizes: [(50,), (100,), (25,25), (75,25), (25,50,25), (10,30,10)] solver: [Stochastic gradient descent, Adam optimizer], activation: [ReLu, Logistic] learning_rate_init: [0.001, 0.01, 0.1]

Table 2: Hyperparameter search grids of all relevant models that grid search selects from

- **f1v.mean**: Average directional-vector maximum Fisher's discriminant ratio.
- **f2.mean**: Average volume of the overlapping region.
- **f3.mean**: Average feature maximum individual efficiency.
- **f4.mean**: Average collective feature efficiency.
- **hubs.mean**: Average hub score.
- **l1.mean**: Average sum of error distance by linear programming.
- **l2.mean**: Average OVO subsets error rate of linear classifier.
- **l3.mean**: Average non-Linearity of a linear classifier.
- **lsc**: Local set average cardinality of a linear classifier.
- **n1**: the fraction of borderline points.
- **n2.mean**: Average ratio of intra and extra class nearest neighbor distance.
- **n3.mean**: Average error rate of the nearest neighbor classifier.
- **n4.mean**: Average non-linearity of the k-NN Classifier.
- **t1**: Fraction of hyperspheres covering data.
- **t2**: The average number of features per dimension.
- **t3**: The average number of PCA dimensions per points.
- **t4**: The ratio of the PCA dimension to the original dimension.

2. Information theory meta-features
   - **attr_conc.mean**: Average concentration coef. of each pair of distinct attributes.
   - **attr_ent.mean**: Average Shannon's entropy for each predictive attribute.

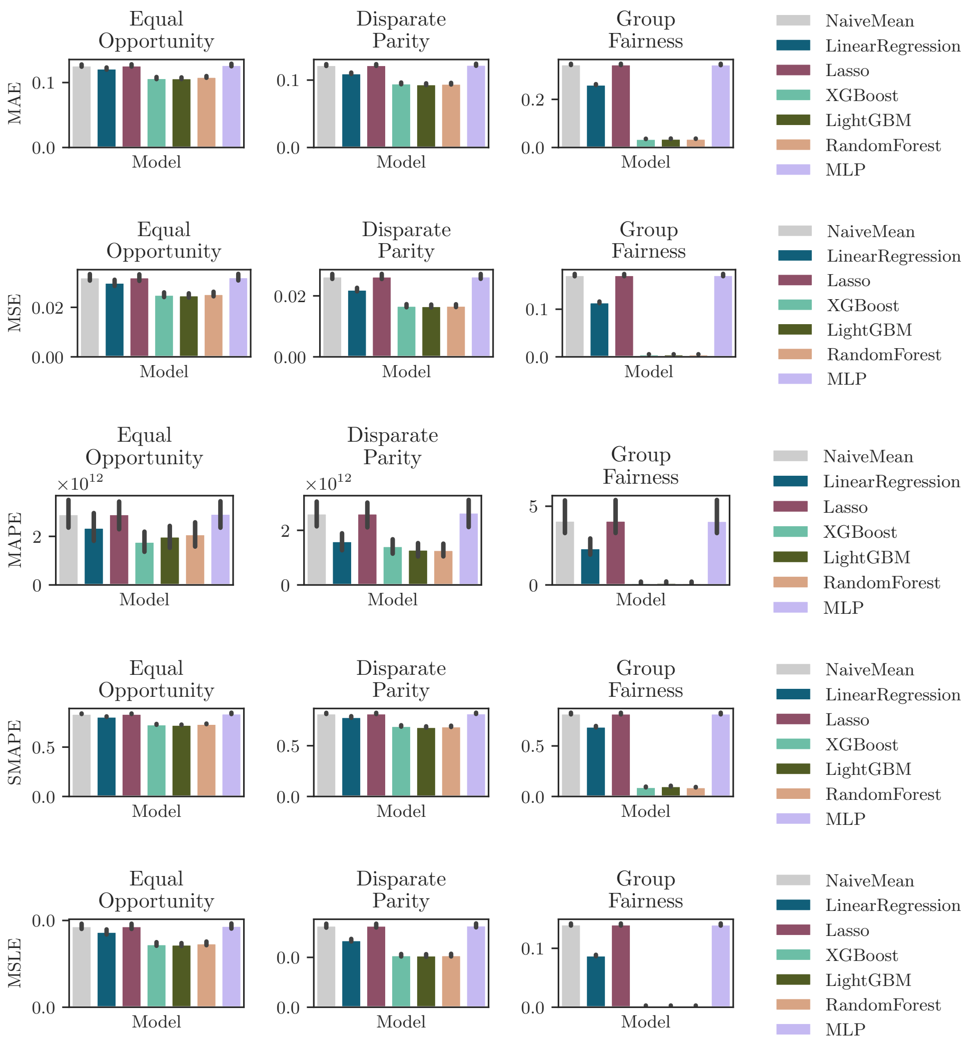

Fig. 3: The performance of the single-target meta-learning models in main text section **4.2** evaluated with MAE, MSE, MAPE, SMAPE and MSLE

- **class_conc.mean**: Average concentration coefficient between each attribute and class.
- **class_ent**: Target attribute Shannon's entropy.
- **eq_num_attr.mean**: Average number of attributes equivalent for a predictive task.
- **joint_ent.mean**: Average joint entropy between each attribute and class.
- **mut_inf.mean**: Average mutual information between each attribute and target.

- **ns_ratio.mean**: Average noisiness of attributes.

3. Landmarking Meta-features
   - **best_node.mean**: Average performance of the best single decision tree landmarkers.
   - **elite_nn.mean**: Average performance of the Elite Nearest Neighbor landmarkers.
   - **linear_discr.mean**: Average performance of the Linear Discriminant classifier landmarkers.
   - **naive_bayes.mean**: Average performance of the Naive Bayes classifier landmarkers.
   - **one_nn.mean**: Average performance of the 1-Nearest Neighbor classifier landmarkers.
   - **random_node.mean**: Average performance of the single decision tree node landmarkers induced by a random attribute.
   - **worst_node.mean**: Average performance of the single decision tree node landmarkers induced by the worst informative attribute.
4. Model-based Meta-features
   - **leaves**: The number of leaf nodes in the trained Decision Tree model.
   - **leaves_branch.mean**: Average size of branches in the trained Decision Tree model.
   - **leaves_corrob.mean**: Average leaves corroboration of the trained Decision Tree model.
   - **leaves_homo.mean**: Average model homogeneity for leaf nodes of the trained Decision Tree model.
   - **leaves_per_class.mean**: Average proportion of leaves per class in trained Decision Tree model
   - **nodes**: The number of non-leaf nodes in the trained Decision Tree model.
   - **nodes_per_attr.mean**: Average ratio of non-leaf nodes per number of attributes in the trained Decision Tree model.
   - **nodes_per_inst.mean**: Average ratio of non-leaf nodes per number of instances in the trained Decision Tree model.
   - **nodes_per_level.mean**: Average ratio of number of non-leaf nodes per tree level in the trained Decision Tree model.
   - **nodes_repeated.mean**: Average number of repeated nodes in the trained Decision Tree model.
   - **tree_depth.mean**: Average depth of nodes in the trained Decision Tree model.
   - **tree_imbalance.mean**: Average tree imbalance for leaf nodes in the trained Decision Tree model.
   - **tree_shape.mean**: Average tree shape for leaf nodes in the trained Decision Tree model.
   - **var_importance.mean**: Average features importance of the trained Decision Tree model for all attributes.

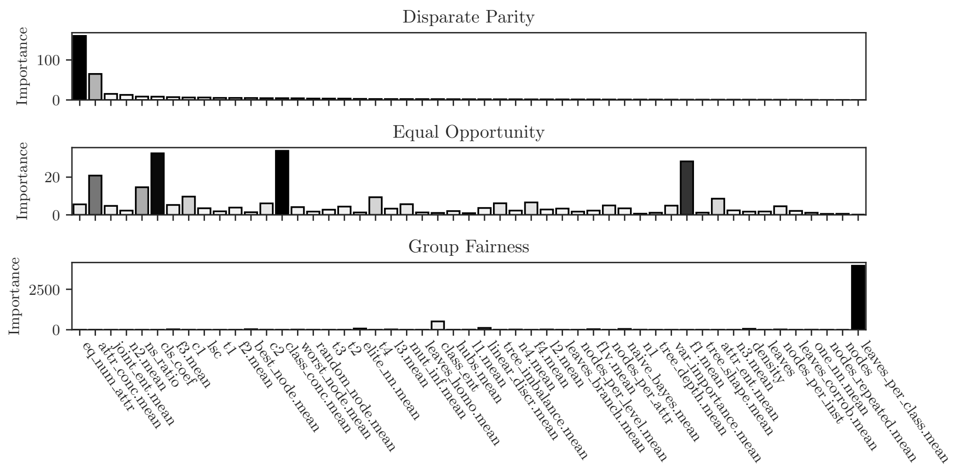

Fig. 4: Feature importance for all meta-features per bias score based on the total gains of the splits using each feature in a single-target LightGBM model

6 Additional Feature Importance Results

In addition to the feature importance results from CLUS presented in the paper, we randomly draw 10×1000 samples from the meta-dataset, train a single-target LightGBM model on each of them for each bias score, and average over the models' feature importance per bias score. Feature importance for a feature here is calculated as the sum of gains of splits using this feature. We provide feature importance results based on the number of splits using the feature in our supplementary materials, alongside the legend of all feature names. Figure 4 shows the results.

We observe that while Disparate Parity and Equal Opportunity use a large number of features, Group Fairness relies heavily on "leaves_per_class", the average number of leaves dedicated to a class. Since Group Fairness seems to be strongly impacted by the number of classes, this feature may be an adequate proxy. If we remove this attribute and repeat the single-target training for Group Fairness, we observe an only 1.2% increase in RMSE and the model uses the features' average Shannon entropy ("class_ent"). Removing this feature as well results in a 2.6% increase in error. The model then relies mostly on "f3", but considers a blend of features. f3 estimates the average efficiency of a single feature in separating the classes [1]; a measure closely related to both other features.

Similarly, Disparate Parity relies strongly on "eq_num_attr", the number of equivalent attributes for a predictive task [2]. Removing it results in a 0.3% increase in RMSE and a focus on "attr_conc", the average concentration coefficient between pairs of features [3]. Removing this feature as well causes a 0.5% increase in error. The model then focuses on "n2", the average ratio of intra/inter class nearest neighbor distance [1], and an overall broader distribution of importance.

Equal Opportunity immediately takes a number of features into account. The most essential ones here are "class_conc", the concentration coefficient between the features

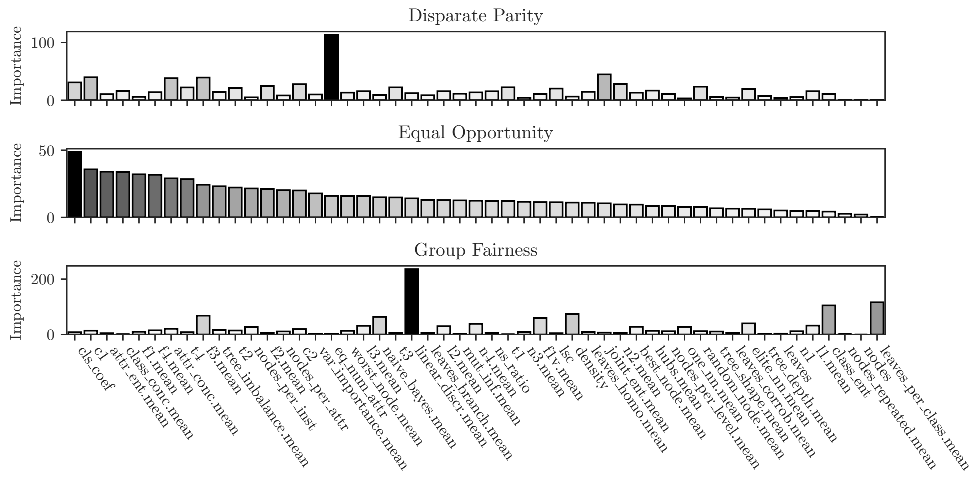

Fig. 5: Feature importance for all meta-feautres per bias score based on the amount of information gained from the single-target LightGBM model by using the feature

and the target [3], "cls_coef", the clustering coefficient (a measure of neighborhood in the dataset's graph representation) [1], and "f1", the maximum Fisher discriminant ratio measuring the overlap of features in different classes [1].

Overall, the least important features are the ones generally describing the shape of the landmarker trees. Figure 5 shows feature importance based on the number of times the feature is used to split the space in single-target LightGBM models. Figures 6 and 7 show the described leave-one-feature-out experiments, an iterative approach of understanding the feature dynamics, where we iteratively remove the most informative feature (based on the cumulative gains of the splits using this feature).

References

[1] Lorena, A.C., Garcia, L.P., Lehmann, J., Souto, M.C., Ho, T.K.: How complex is your classification problem? a survey on measuring classification complexity. ACM Computing Surveys **52**, 1–34 (2019)

[2] Michie, D., Spiegelhalter, D.J., Taylor, C.C., Campbell, J.: Machine Learning, Neural and Statistical Classification. Ellis Horwood, USA (1995)

[3] Kalousis, A., Hilario, M.: Model selection via meta-learning: a comparative study. In: Proceedings 12th IEEE Internationals Conference on Tools with Artificial Intelligence, pp. 406–413 (2000)

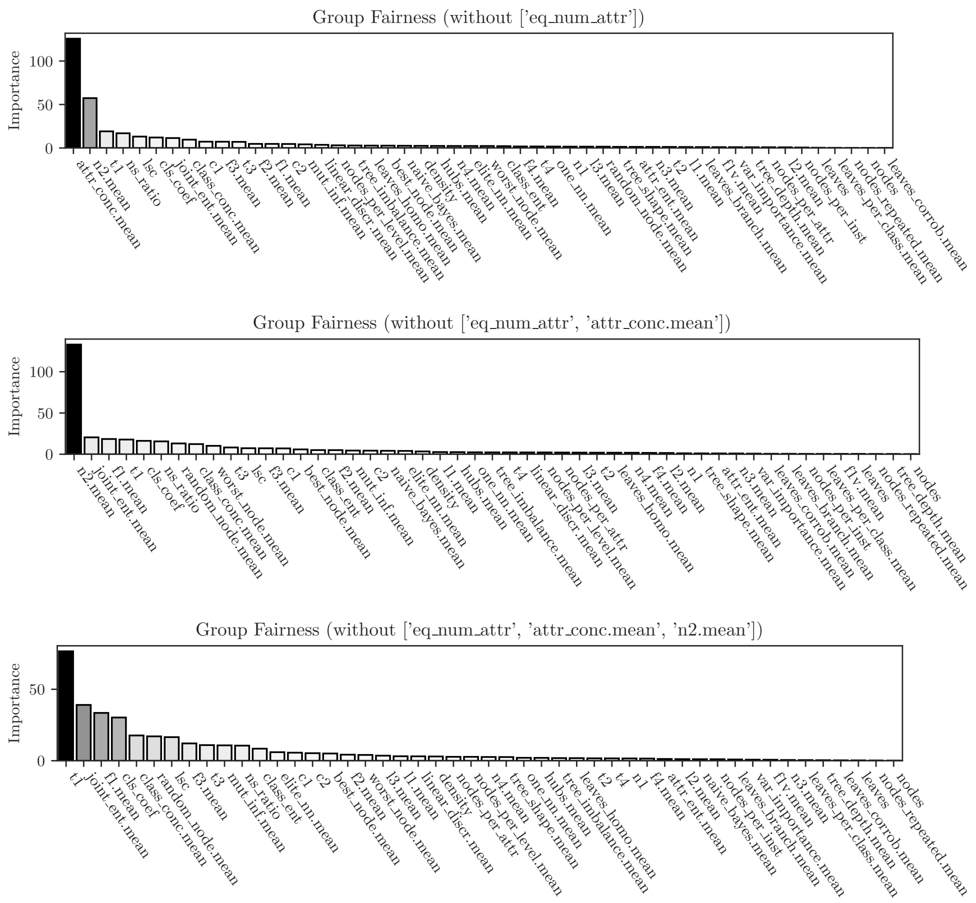

Fig. 6: Feature importance (based on gain) for iterative feature removal experiment on Disparate Parity

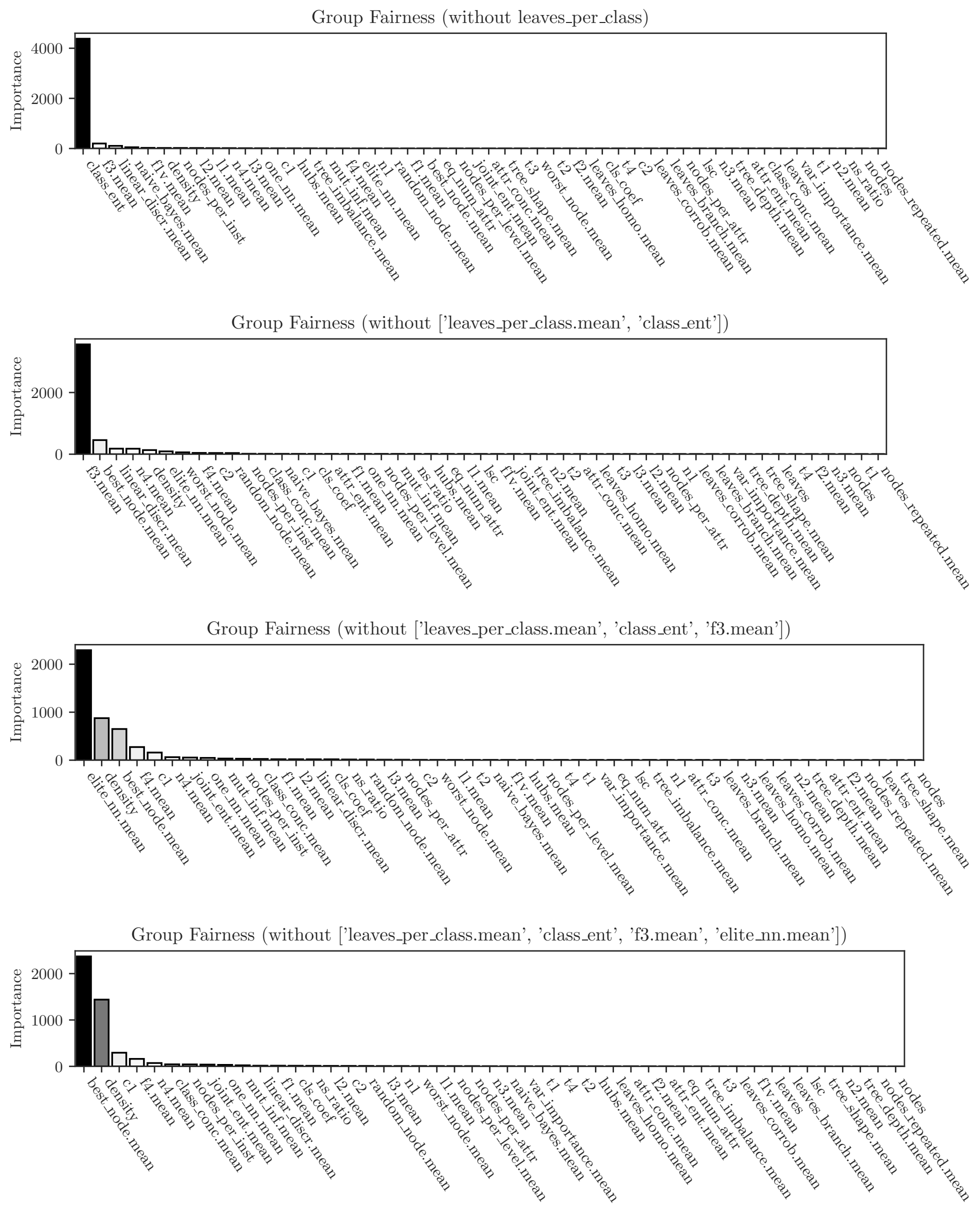

Fig. 7: Feature importance (based on gain) for iterative feature removal experiment on Group Fairness