Upon Detection and Mitigation of Dataset Bias

This project focuses on detecting and mitigating bias in tabular and image datasets. The libraries that are required to run the code include: matplotlib, pytorch, numpy, pandas, visdom (only for tracking training progress) and scikit-learn. These can be installed through pip3:

```
pip3 install <library_name>
```

This document will serve as a guide to understand most important scripts.

Authors

- Gabriel Hayat

Datasets

In the `Datasets` folder, two datasets are relevant for this project:

- Adult dataset: This dataset is tabular. It involves using personal details to predict whether an individual's yearly income exceeds $50,000 per year. The features used for predictiona include the age, the gender, the education level, the race etc ... The dataset contains 48'842 samples. When handling this dataset, we are going to set *gender* as the sensitive attribute and will extend this set with the *race* attribute when looking at subgroup fairness.
- basket_volley: This sport dataset is an image dataset composed of 1643 samples, split into two classes, namely a *basketball* and *volleyball* class. The classes are balanced (i.e. number of samples in the two classes are similar). The images consist of in-game pictures, portraits and team pictures in their respective discipline. For the sake of exploring fairness, we manually divided the classes according to the jersey color of the player(s) represented in the images, which we use as a sensitive attribute when handling this dataset.

Details about how these datasets are preprocessed and prepared for training can be found in the report.

Clustering

This folder refers to the bias detection section of the project. The goal is to detect whether the dataset is biased against certain sub-population(s). This folder refers to the bias detection section of the project. The goal is to detect whether the dataset is biased against certain sub-population(s).

- **unsupervised_clustering.ipynb** : This notebook aims at clustering the dataset to expose potential bias. It uses a small convolutional network to find embeddings of every sample of the dataset, before applying PCA and using the k-means algorithm with a customized distance metric to yield a clustering of the dataset. This notebook can be run sequencially, cell by cell. (see report for more details).
- **semi-supervised_clustering.ipynb** : This notebook aims at clustering the dataset using a classifier that is trained on a small portion of the data. It uses a Residual Network as a classifier. After being trained, it is used to predict the sensitive attribute of the trasining data and allocates each sample to its corresponding cluster based on its prediction and the sample's class. This notebook can be run sequencially, cell by cell. (see report for more details).
- **K-means (basket_volley).ipynb** : This notebook explores the basket_volley dataset and investigates new approaches, it can be ignored.
- **K-means (doctor_nurse).ipynb** : This notebook explores the doctor_nurse dataset, which is not talked about in this project. This notebook can be ignored.

Logistic Regression

This folder deals with the Adult tabular dataset described above. It contains scripts that train and measure the performance of baselines as well as our reweigthing algorithms (see report for description). The main scripts are listed here, as well as commands to run them:

- **main.py** : This is the main script of the folder. Its behavior will depend on the arguments passed.

```
python3 main.py -label_column={LABEL_COL} -protect_columns={PROTECT_COLS} -mode={MODE} -start_epoch={START_EPO
```

where:

1. label_column=<label_column>: the name of the target column
2. protect_columns=<protect_columns> (separated by a comma, no space)
   - gender - male vs female (protected)
   - race_White - white vs non-white (protected)
3. mode=<mode>
   - 0: Model is trained on bias dataset as it is, no reweighting
   - 1: Model is trained on customed dataset, where each sample is reweighted as to have the same number of minority and majority samples per class (only works when there is one protected column)
   - 2: Model is trained on customed dataset, where weights of each cluster/sample is dynamically updated
4. update=<update> (This parameter is only relevant when in MODE 2)
   - cluster: each cluster has a weight
   - sample: each sample has a weight
5. weights_init = <weights_init> (This parameter is only relevant in MODE 2)
   - 0: the cluster/sample weights are initialized with unit weight
   - 1: the cluster/sample weights are initialized with weights from MODE 1 (only works when there is one protected column)
6. start_epoch=<start_epoch> : the epoch to start from if the model has already been trained
7. num_epoch=<num_epoch>
8. id=<id>: the id of the trained model
9. num_trials=<num_trials>: number of times to repeat the training process
10. num_proxies= <num_proxies>: number of features to removed that are most correlated with the label
11. file_path=<file_path>: the file path of the preprocessed data
12. verbose=<verbose>
13. lr=<lr>: the learning rate
14. update_lr=<update_lr> : the learing rate of the weight updates
15. batch_size=
16. . balance=<balance>
    - 0: The training set and test set is not rebalanced in any way
    - 1: The training set is rebalanced in terms of labels and the test set is rebalanced in terms of labels and groups/subgroups

When the training procedure ends, the checkpoint as well as some evaluation statistics will be saved at the path: `./Case_{MODE + 1}/checkpoints/model_ep_{START_EPOCH + NUM_EPOCH}/Run_{ID}/` .

- **base_rate_generator.py** : This first splits the dataset into the majority and minority groups, and trains a logistic regression model on each of them. It then records the difference between the performances of the two classifiers. See details about the base rate analysis in the report.

```
python3 base_rate_generator.py -label_column={LABEL_COL} -protect_columns={PROTECT_COLS} -num_epoch={NUM_EPOCH
```

where

1. label_column=<label_column>: the name of the target column
2. protect_columns=<protect_columns> (separated by a comma, no space)
   - gender - male vs female (protected)
   - race_White - white vs non-white (protected)
3. num_epoch=<num_epoch>
4. id=<id>: the id of the trained model
5. num_trials=<num_trials>: number of times to repeat the training process
6. num_proxies= <num_proxies>: number of features to remove that are most correlated with label
7. batch_size= <batch_size>: controls the batch size of the classifiers.
8. keep=
   - The proportion to keep when filtering the majority and minority groups (must be]0,1])
9. filter_maj --filter_min
   - 1: filters the group to improve model predictions
   - 0: does not filter the group
   - -1: filters the group to worsen model predictions

When the training procedure ends, the checkpoint as well as some evaluation statistics will be saved at the path: `./base_rate_models/checkpoints/model_ep_{NUM_EPOCH}/Run_{ID}/` .

- The other scripts of the folder are briefly mentioned below:

File	Description
adversial_model.py	Baseline model, see: [1].
calibrated_eq_odds_postprocessing.py	Baseline model: see [2]
disparate_impact_remover.py	Baseline model, see [3]
reject_option_classifier.py	Baseline mode, see [4]
evaluate.py	Evaluates a trained model against multiple fairness metrics
fairness_metrics.py	Contains the fairness metrics defined in the report
load_dataset.py	Preprocesses the dataset and prepares it for training
logistic_regression_model.py	Contains the model defintion, the training and evaluation methods

[1] Brian Hu Zhang, Blake Lemoine, Margaret Mitchell. *Mitigating Unwanted Biases with Adversarial Learning*, Stanford University, Google Mountain View, CA (2018)

[2] G. Pleiss, M. Raghavan, F. Wu, J. Kleinberg, andK. Q. Weinberger, *On Fairness and Calibration*, Conference on Neural Information Processing Systems, 2017

[3] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, *Certifying and removing disparate impact*, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015.

[4] [10] F. Kamiran, A. Karim, and X. Zhang, *Decision Theory for Discrimination-Aware Classification*, IEEE International Conference on Data Mining, 2012.

Resnet

This folder deals with the basket_volley image dataset described above. It contains scripts that train and measure the performance of baselines as well as our reweghting algorithms. Note that for images, the reweighting model is a

Residual Neural Network model, where we freeze all layers except from the last two, which we train on our dataset. The main scripts are listed here, as well as commands to run them:

- **Case_1+2.py** : This script takes care of the two following cases:
  - The resnet is trained on the dataset and its performance (accuracy vs fairness trade-off) is computed.
  - The resnet is trained on a reweighted dataset, where the minority class is reweighted by an input weight defined by the user

  ```
  python3 Case_1+2.py -w_protected={W_PROTECTED} -bias={BIAS} -val_mode={VAL_MODE} -start_epoch={START_EPOCH
  ```

where:

1. w_protected=<w_protected>: weight with which to reweight every sample of the minority class
2. bias=<bias>: the version of the dataset to use, i.e. bias ranges from 0.5 to 0.8 and represents the ratio of `majority : minority` in each class.
3. start_epoch=<start_epoch> : the epoch to start from if the model has already been trained
4. num_epoch=<num_epoch>
5. val_mode=<val_mode>: whether to train the model in validation mode
6. id=<id>: the id of the trained model
7. num_trials=<num_trials>: number of times to repeat the training process
8. visom=<visdom>: boolean that determines whether to plot the training process of the model (visdom diplays the plots on an external server, see documentation: https://github.com/fossasia/visdom)

When the training procedure ends, the checkpoint as well as some evaluation statistics will be saved at the path: `./("Case_2/" if W_PROTECTED != 1 else "Case_1/") + "checkpoints/" + ( "w_val" if VAL_MODE else "w.o_val") + f"/Bias_{BIAS}/model_ep_{START_EPOCH + NUM_EPOCH}/Run_{ID}/` .

- **data_reweghting.py** : This scripts trains and evaluates our algorithms on the image dataset.

```
python3 data_reweighting.py -w_protected={W_PROTECTED} -bias={BIAS} -val_mode={VAL_MODE} -start_epoch={START_E
```

where:

1. w_protected=<w_protected>: weight with which to reweight every sample of the minority class
2. bias=<bias>: the version of the dataset to use, i.e. bias ranges from 0.5 to 0.8 and represents the ratio of `majority : minority` in each class.
3. label_column=<label_column>: the name of the target column start_epoch=<start_epoch> : the epoch to start from if the model has already been trained
4. num_epoch=<num_epoch>
5. val_mode=<val_mode>: whether to train the model in validation mode
6. id=<id>: the id of the trained model
7. num_trials=<num_trials>: number of times to repeat the training process
8. update=<update>: this argument decides which algorithm to use (see report for more detailed explainations):
   - cluster: each cluster has a weight
   - sample: each sample has a weight
   - individual: each sample is treated as an independent individual
9. update_lr=<update_lr>: the weight update learning rate of the algorithm
10. clusters=<clusters>: this parameter should be set when using customized clusters: it should be the name of a python dictionary mapping each sample to its cluster

When the training procedure ends, the checkpoint as well as some evaluation statistics will be saved at the path: `./"Reweighting/checkpoints/" + ("cluster_update/" if UPDATE == "cluster" else ("sample_update/" if UPDATE == "sample" else "individual_update/")) + ("w_val" if VAL_MODE else "w.o_val") + f"/Bias_{BIAS}/model_ep_{START_EPOCH + NUM_EPOCH}/Run_{ID}/` .

- The other scripts of the folder are briefly mentioned below:

File	Description
Case_3.py	Trains the model on a randomized reweighted dataset, this class is depreciated.
fairness_metrics.py	Contains the fairness metrics defined in the report
my\ImageFolder.py	Preprocesses the datastet, splits it in terms of clusters and prepares it for training
model.py	Contains the Resnet as well as methods to train and evaluate it
Result.py	Evaluates a trained model against multiple fairness metrics