fairensics

Niki Kilbertus, Danilo Brajovic

Jan 29, 2020

CONTENTS

Fairensics is a python library to discover and mitigate biases in machine learning models and datasets. The best location to learn about fairensics are the jupyter notebooks.

Fairensics is based on AIF360 and provides compatible versions of the fairness methods found here.

CHAPTER

ONE

INSTALLATION

1.1 Basic installation instructions

Fairensics was tested with Python 3.6 on OS X and Linux. We recommend using *pip* for installation within an virtualenv.

1. (Optionally) create a virtual environment.

```
python3 -m venv fairensics-env
source fairensics-env/bin/activate
```

2. Install the package.

```
pip install fairensics
```

CHAPTER

TWO

API REFERENCE

2.1 API

2.1.1 Data & Plotting

class fairensics.data.decision_boundary.**DecisionBoundary**(*colors=('k', 'c', 'm', 'b', 'g', 'r', 'y')*, *downsampler=PCA(copy=True, iterated_power='auto', n_components=2, random_state=None, svd_solver='auto', tol=0.0, whiten=False)*)

Class for plotting decision boundaries against two axes.

The data may be down sampled to two dimensions before plotting. The decision boundary plots are generated using a mesh grid and the following procedure:

1. If necessary, the data is down-sampled to two dimensions
2. Min and maximum values for each axis are extracted
3. A mesh grid is created
4. If necessary, the mesh grid is up-sampled again
5. Predictions are made on the mesh grid
6. Predictions are plotted against the maybe down sampled axis

TODO: add option to scale data to [0,1]

__init__(*colors=('k', 'c', 'm', 'b', 'g', 'r', 'y')*, *downsampler=PCA(copy=True, iterated_power='auto', n_components=2, random_state=None, svd_solver='auto', tol=0.0, whiten=False)*)

Parameters

- **colors** – iterator over possible colors for the decision boundaries
- **downsampler** – function to down sample data points must implement 'fit_transform' and 'inverse_transform' methods

add_boundary(*dataset*, *clf*, *label=''*, *only_unprotected=True*, *num_points=100*, *cmap=None*)

Adds decision boundary to the current plot.

If the data set is two dimensional, the boundary is directly plotted using a mesh grid. Otherwise, a mesh gird is generated on the down-sampled points and up-sampled again for prediction.

Parameters

- **dataset** (*BinaryLabelDataset*) – the labeled data set.
- **clf** (*object*) – the classifier object (must implement a predict function).
- **label** (*str*) – the label for the decision boundary.
- **only_unprotected** (*bool*) – if true, the classifier only uses the unprotected attributes.
- **num_points** (*int*) – number of points in mesh grid.
- **cmap** (*str*) – colormap from matplotlib. If provided background of the plot is colored.

scatter(*dataset*, *protected_attribute_ind=0*, *only_unprotected=True*, *num_to_draw=100*)

Scatter plot the points in dataset.

Protected and unprotected individuals and positive and negative label are distinguished. Only one protected attribute is considered for plotting.

Parameters

- **dataset** (*BinaryLabelDataset*) – data set to plot.
- **protected_attribute_ind** (*int*) – index of the protected attribute to consider.
- **only_unprotected** (*bool*) – if true, the classifier only uses the unprotected attributes.
- **num_to_draw** (*int*) – number of points to draw.

static show(*title=''*, *xlabel=''*, *ylabel=''*)

Shows the plot

class fairensics.data.synthetic_dataset.**SyntheticDataset**(*n_samples=1000*, *label_name='label'*, *feature_one_name='feature_1'*, *feature_two_name='feature_2'*, *favorable_label=1*, *unfavorable_label=0*, *protected_attribute_name='protected_attribute'*, *privileged_class=1*, *unprivileged_class=0*, *sd=1122334455*, *mu_1=(2, 2)*, *sigma_1=((5, 1), (1, 5))*, *mu_2=(-2, -2)*, *sigma_2=((10, 1), (1, 3))*, *initial_discrimination=4.0*)

Synthetic data set with two features and one protected attribute.

The data set is randomly generated from two gaussians each time. Both protected attribute and label are binary and features are numerical.

__init__(*n_samples=1000*, *label_name='label'*, *feature_one_name='feature_1'*, *feature_two_name='feature_2'*, *favorable_label=1*, *unfavorable_label=0*, *protected_attribute_name='protected_attribute'*, *privileged_class=1*, *unprivileged_class=0*, *sd=1122334455*, *mu_1=(2, 2)*, *sigma_1=((5, 1), (1, 5))*, *mu_2=(-2, -2)*, *sigma_2=((10, 1), (1, 3))*, *initial_discrimination=4.0*)

Parameters

- **n_samples** (*int*) – the number of samples to generate
- **label_name** (*str*) – name of the column storing the target variable
- **feature_one_name** (*str*) – name of the first unprotected feature
- **feature_two_name** (*str*) – name of the second unprotected feature
- **favorable_label** (*int*) – label considered positive
- **unfavorable_label** (*int*) – label considered negative
- **protected_attribute_name** (*str*) – the name of the protected attribute
- **privileged_class** (*int*) – class of protected attribute considered positive
- **unprivileged_class** (*int*) – class of protected attribute considered negative
- **sd** (*int*) – seed for random generator
- **mu_1** (*float, float*) – mean of positive group cluster
- **sigma_1** (*(float, float), (float, float)*) – covariance of positive group cluster
- **mu_2** (*float, float*) – mean of negative group cluster
- **sigma_2** (*(float, float), (float, float)*) – covariance of negative group cluster
- **initial_discrimination** (*float*) – initial discrimination factor

plot(*num_to_draw=200*)
Plot subsample of data with unprotected features on x and y axis.

2.1.2 Modeling

class fairensics.methods.disparate_impact.**AccurateDisparateImpact**(*loss_function='logreg', warn=True*)
Minimize loss subject to fairness constraints.

Loss "L" defines whether a logistic regression or a liner SVM is trained.

Minimize L(w)

Subject to cov(sensitive_attributes, true_labels, predictions) < sensitive_attrs_to_cov_thresh

Where: predictions: the distance to the decision boundary

__init__(*loss_function='logreg', warn=True*)
Args: loss_function (str): loss function string from utils.LossFunctions. warn (bool): if true, warnings are raised on certain bounds.

fit(*dataset, sensitive_attrs_to_cov_thresh=0, sensitive_attributes=None*)
Fit the model.

Parameters

- **dataset** – AIF360 data set
- **sensitive_attrs_to_cov_thresh** (*float or dict*) – dictionary as returned by _get_cov_thresh_dict(). If a single float is passed the dict is generated using the _get_cov_thresh_dict() method.
- **sensitive_attributes** (*list(str)*) – names of protected attributes to apply constraints to.

class fairensics.methods.disparate_impact.**FairDisparateImpact**(*loss_function='logreg'*, *warn=True*)

Minimize disparate impact subject to accuracy constraints.

Loss "L" defines whether a logistic regression or a liner svm is trained.

Minimize cov(sensitive_attributes, predictions)

Subject to L(w) <= (1-gamma)L(w*)

Where L(w*): is the loss of the unconstrained classifier predictions: the distance to the decision boundary

__init__(*loss_function='logreg'*, *warn=True*)

Args: loss_function (str): loss function string from utils.LossFunctions. warn (bool): if true, warnings are raised on certain bounds.

fit(*dataset*, *sensitive_attributes=None*, *sep_constraint=False*, *gamma=0*)

Fits the model.

Parameters

- **dataset** – AIF360 data set.
- **sensitive_attributes** (*list(str)*) – names of protected attributes to apply constraints to.
- **sep_constraint** (*bool*) – apply fine grained accuracy constraint.
- **gamma** (*float*) – trade off for accuracy for sep_constraint.

class fairensics.methods.disparate_mistreatment.**DisparateMistreatment**(*loss_function='logreg'*, *constraint_type=None*, *take_initial_sol=True*, *warn=True*, *tau=0.005*, *mu=1.2*, *EPS=1e-06*, *max_iter=100*, *max_iter_dccp=50*)

Disparate mistreatment free classifier. Loss "L" defines whether a logistic regression or a liner svm is trained.

Minimize L(w)

Subject to cov(sensitive_attributes, predictions) < sensitive_attrs_to_cov_thresh

Where predictions: the distance to the decision boundary

Example

https://github.com/nikikilbertus/fairensics/blob/master/examples/2_2_fair-classification-mistreatment-example.ipynb

__init__(*loss_function='logreg'*, *constraint_type=None*, *take_initial_sol=True*, *warn=True*, *tau=0.005*, *mu=1.2*, *EPS=1e-06*, *max_iter=100*, *max_iter_dccp=50*)

Parameters

- **loss_function** (*str*) – name of loss function defined in utils
- **constraint_type** (*str*) – one of the values in _CONS_TYPE
- **take_initial_sol** (*bool*) –

- **warn** (*bool*) – if true, warnings are raised on certain bounds
- **mu, EPS, max_iter, max_iter_dccp** (*tau,*) – solver related parameters

fit (*dataset*, *sensitive_attrs_to_cov_thresh=0*)
Fits the model.

Parameters

- **dataset** – AIF360 data set
- **sensitive_attrs_to_cov_thresh** (*dict or float*) – covariance between sensitive attribute and decision boundary

predict (*dataset*)
Make predictions.

Parameters **dataset** – AIF360 data set

Returns either AIF360 data set or np.array if dataset is also np.array

class fairensics.methods.preferential_fairness.**PreferentialFairness**(*loss_function='logreg'*, *constraint_type=None*, *train_multiple=False*, *lam=None*, *warn=True*, *tau=0.5*, *mu=1.2*, *EPS=0.0001*, *max_iter=100*, *max_iter_dccp=50*)
Train separate classifier clf_z for each group of protected attribute z. Loss "L" defines whether a logistic regression or a liner svm is trained.

Minimize L(w)

Subject to sum(predictions_z) > sum(predictions_z')

Where predictions_z are the predictions using group zs classifier clf_z predictions_z' are the predictions using group z's classifier clf_z'

Example

https://github.com/nikikilbertus/fairensics/blob/master/examples/2_3_fair-classification-preferential-fairness-example.ipynb

__init__ (*loss_function='logreg'*, *constraint_type=None*, *train_multiple=False*, *lam=None*, *warn=True*, *tau=0.5*, *mu=1.2*, *EPS=0.0001*, *max_iter=100*, *max_iter_dccp=50*)

Parameters

- **loss_function** (*str*) – name of loss function defined in utils.
- **constraint_type** (*str*) – one of the values in _CONS_TYPE.
- **train_multiple** (*bool*) – if true, a classifier for each group of protected attribute is trained
- **lam** (*dict, optional*) – ...
- **warn** (*bool*) – if true, warnings are raised on certain bounds.
- **mu, EPS, max_iter, max_iter_dccp** (*tau,*) – solver related parameters.

fit (*dataset*, *s_val_to_cons_sum=None*, *prot_attr_ind=0*)
Fits the model.

Parameters

- **dataset** – AIF360 data set.
- **s_val_to_cons_sum** (*dict*) – the ramp approximation, only needed for _constraint_type 1 and 3.
- **prot_attr_ind** (*int*) – index of the protected feature to apply constraints to.

predict (*dataset*)
Make predictions.

Parameters **dataset** – either AIF360 data set or np.ndarray.

Returns either AIF360 data set or np.ndarray if dataset is a np.ndarray.

class fairensics.methods.fairness_warnings.**FairnessBoundsWarning**(*raw_dataset*, *predicted_dataset*, *privileged_groups=None*, *unprivileged_groups=None*)
Raise warnings if classifier misses specified fairness bounds.

Bounds are checked using AIF360s classification metric if the specified bound is not None.

DISPARATE_IMPACT_RATIO_BOUND = 0.8

EO_DIFFERENCE_BOUND = 0.1

ERROR_DIFFERENCE_BOUND = None

ERROR_RATIO_BOUND = 0.8

FNR_DIFFERENCE_BOUND = None

FNR_RATIO_BOUND = 0.8

FPR_DIFFERENCE_BOUND = None

FPR_RATIO_BOUND = 0.8

__init__ (*raw_dataset*, *predicted_dataset*, *privileged_groups=None*, *unprivileged_groups=None*)

Parameters

- **raw_dataset** (*BinaryLabelDataset*) – Dataset with ground-truth labels.
- **predicted_dataset** (*BinaryLabelDataset*) – Dataset after predictions.
- **privileged_groups** (*list(dict)*) – Privileged groups. Format is a list of *dicts* where the keys are *protected_attribute_names* and the values are values in *protected_attributes*. Each *dict* element describes a single group.
- **unprivileged_groups** (*list(dict)*) – Unprivileged groups. Same format as privileged_groups.

check_bounds()
Run methods checking each bound.

class fairensics.methods.fairness_warnings.**DataSetSkewedWarning**(*dataset*)
Raise warning if dataset is skewed with respect to protected attributes.

Checks are only executed, if the specified bounds are not None.

CLASS_LABEL_FRACTION = 0.4

POSITIVE_NEGATIVE_CLASS_FRACTION = 0.4

POSITIVE_NEGATIVE_LABEL_FRACTION = 0.4

__init__(*dataset*)

Parameters **dataset** (*BinaryLabelDataset*) – the ground truth data set.

check_dataset()
Call methods checking bounds if bounds are specified.

class fairensics.methods.utils.**LossFunctions**
Loss functions for fair-classification.

This class stores implementations of loss functions used in fair-classification. The functions can be accessed using the get_loss_function() methods passing loss function names either as numpy or cvxpy implementation.

LOSS_NAMES = ['logreg', 'logreg_l1', 'logreg_l2', 'svm_linear']

NAME_LOG_REG = 'logreg'

NAME_LOG_REG_L1 = 'logreg_l1'

NAME_LOG_REG_L2 = 'logreg_l2'

NAME_SVM_LOSS = 'svm_linear'

static cvxpy_hinge_loss(*w*, *X*, *y*, *num_points=None*)
CVXPY implementation of hinge loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **num_points** (*int*) – number of points in X (corresponds to the first dimension of X "n", but some methods pass a different value for scaling).

Returns the loss.

Return type (float)

static cvxpy_logistic_loss(*w*, *X*, *y*, *num_points=None*)
CVXPY implementation of logistic loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **num_points** (*int*) – number of points in X (first dimension of X "n_samples", but some methods pass a different value for scaling).

Returns the loss.

Return type (float)

static cvxpy_logistic_loss_l1(*w*, *X*, *y*, *lam=None*, *num_points=None*)
CVXPY implementation of L1 regularized logistic loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **lam** (*float*) – regularization parameter.
- **num_points** (*int*) – number of points in X (corresponds to the first dimension of X "n", but some methods pass a different value for scaling).

Returns the loss.

Return type (float)

static cvxpy_logistic_loss_l2(*w*, *X*, *y*, *lam=None*, *num_points=None*)
CVXPY implementation of L2 regularized logistic loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **lam** (*float*) – regularization parameter.
- **num_points** (*int*) – number of points in X (corresponds to the first dimension of X "n", but some methods pass a different value for scaling).

Returns the loss.

Return type (float)

static get_cvxpy_loss_function(*loss_name*)
Return cvxpy loss function for loss_name.

static get_loss_function(*loss_name*)
Return loss function for loss_name.

static hinge_loss(*w*, *X*, *y*)
Numpy implementation of hinge loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).

Returns the loss.

Return type (float)

static log_logistic(*X*)
Log_logistic from scikit-learn source code. Source link below.

Compute the log of the logistic function, `log(1 / (1 + e ** -x))`. Source code at: https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/utils/extmath.py

Parameters **X** (*array-like*) – shape (M, N) Argument to the logistic function

Returns

shape (M, N) Log of the logistic function at every point in x

Return type out (np.ndarray)

static logistic_loss(*w*, *X*, *y*, *return_arr=False*)
Numpy implementation of logistic loss.

This function is used from scikit-learn source code

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **return_arr** (*bool*) – if true, an array is returned otherwise the sum of the array

Returns the loss.

Return type (float or list(float))

static logistic_loss_l1_reg(*w*, *X*, *y*, *lam=None*)
Numpy implementation of L1 regularized logistic loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **lam** (*float*) – regularization parameter.

Returns the loss.

Return type (float)

static logistic_loss_l2_reg(*w*, *X*, *y*, *lam=None*)
Numpy implementation of L2 regularized logistic loss.

Parameters

- **w** (*np.ndarray*) – 1D, the weight matrix with shape (n_features,).
- **X** (*np.ndarray*) – 2D, the features with shape (n_samples, n_features)
- **y** (*np.ndarray*) – 1D, the true labels with shape (n_samples,).
- **lam** (*float*) – regularization parameter.

Returns the loss.

Return type (float)

fairensics.methods.utils.**get_one_hot_encoding**(*arr*)
Returns one hot encoding of array arr.

Parameters **arr** (*np.ndarray*) – 1D array with int values.

Returns Tuple consisting of out_arr (np.ndarray) one-hot encoded matrix and index_dict (dict) dictionary original_val -> column in encoded matrix.

fairensics.methods.utils.**add_intercept**(*x*)
Adds intercept (column of ones) to X.

fairensics.methods.utils.**get_protected_attributes_dict**(*names*, *attributes*)
Returns dictionary of protected attributes.

The dictionary has the form: {"s1": [...], "s2": [...], ... } Key "sI" is the sensitive feature name, and [...] the 1D array holding the sensitive feature.

Parameters

- **names** (*list(str)*) – names of the attributes in attributes.
- **attributes** (*np.ndarray*) – 2D array of the sensitive features.

Returns {"s1": [attributes[:, 1]], "s2":[attributes[:, 2]], ... }

Return type (dict)

2.1.3 Utilities

Utility functions.

fairensics.fairensics_utils.**get_unprotected_attributes**(*dataset*)
Returns unprotected features from data set.

Parameters **dataset** (*StructuredDataset*) – data set with features, protected features and labels.

Returns (np.ndarray) of unprotected features only

PYTHON MODULE INDEX

f

INDEX

Symbols

A

C

D

E

F

G

H

L

N

P

S