TransparentAI

Release 0.1.0

Jul 30, 2020

TransparentAI

This Python library was developed by Nathan Lauga a french Data Scientist.

If you find spelling or grammar mistakes that hurt your eyes, I apologize in advance. But do not hesitate to report them on the issues GitHub page from the library here : https://github.com/Nathanlauga/transparentai/issues

CHAPTER 1

Quick start

Install the library with pypi :

```
>>> pip install transparentai
```

1.1 Installation

Install TransparentAI with PyPI :

```
>>> pip install transparentai
```

Or by cloning GitHub repository :

```
>>> git clone https://github.com/Nathanlauga/transparentai.git
>>> cd transparentai
>>> python setup.py install
```

1.2 Getting started with TransparentAI

This page will show you some code to start with the TransparentAI library.

In this section I created a binary classifier based on Adult dataset. The following variables will be used :

variable	description
data	Adult dataset as DataFrame
clf	Classifier model
y_true	True labels for train set
y_true_valid	True labels for valid set
y_pred	Predictions labels for train set
y_pred_valid	Predictions labels for valid set
df_valid	Dataframe for valid set
X_train	Features for train set
X_valid	Features for valid set

1.2.1 Is my model biased ?

```
>>> privileged_group = {
    # For gender attribute Male peoples are considered to be privileged
    'gender':['Male'],
    # For marital-status attribute Married peoples are considered to be privileged
    'marital-status': lambda x: 'Married' in x,
    # For race attribute White peoples are considered to be privileged
    'race':['White']
}
```

```
>>> from transparentai import fairness
>>> fairness.model_bias(y_true_valid, y_pred_valid, df_valid, privileged_group)
{
    "gender": {
        "statistical_parity_difference": -0.07283528047741014,
        "disparate_impact": 0.4032473042703101,
        "equal_opportunity_difference": -0.04900038770381182,
        "average_odds_difference": -0.026173142849183567
    },
    "marital-status": {
        "statistical_parity_difference": -0.11667610209029305,
        "disparate_impact": 0.27371312304160633,
        "equal_opportunity_difference": 0.08345535064884008,
        "average_odds_difference": 0.03867329810319946
    },
    "race": {
        "statistical_parity_difference": -0.0420778376239787,
        "disparate_impact": 0.5964166117990216,
        "equal_opportunity_difference": -0.0004408949904296522,
        "average_odds_difference": -0.002870373184105955
    }
}
```

This metrics can be not easy to understand so you can use the returns_text=True so that you can get ths insight :

```
>>> fairness_txt = fairness.model_bias(y_true_valid, y_pred_valid, df_valid,␣
↪privileged_group, returns_text=True)
>>> print(fairness_txt['gender'])
The privileged group is predicted with the positive output 7.28% more often than the␣
↪unprivileged group. This is considered to be fair.
The privileged group is predicted with the positive output 2.48 times more often than␣
↪the unprivileged group. This is considered to be not fair.
```

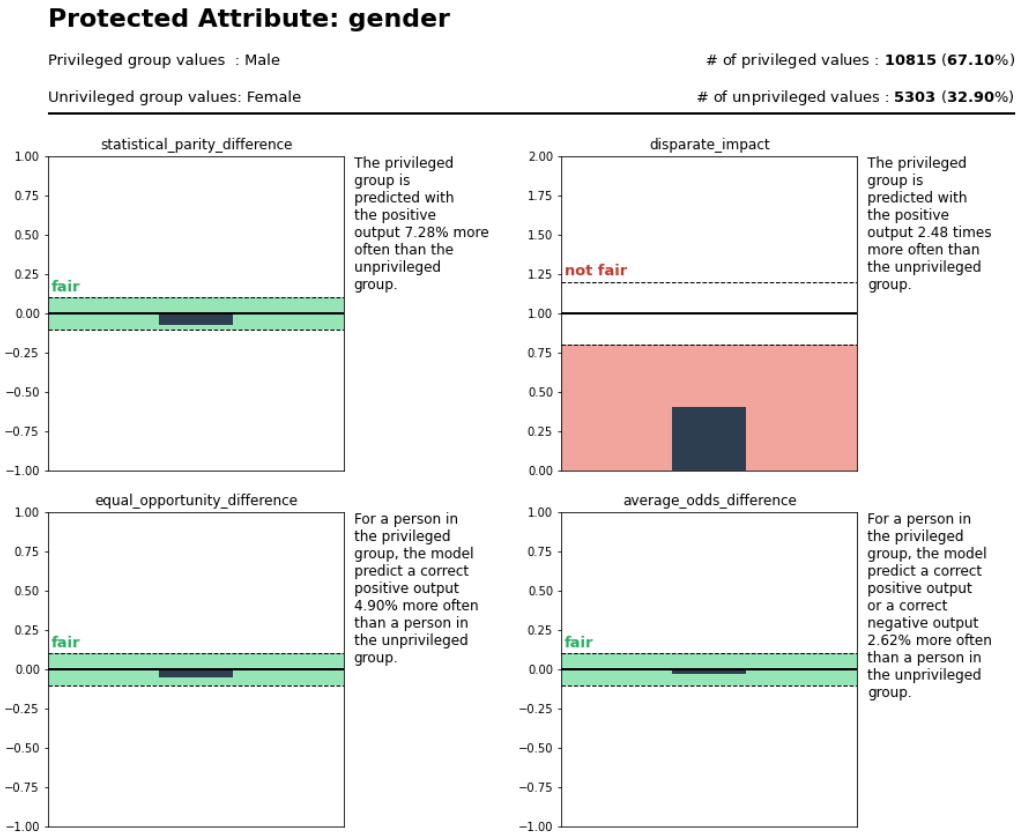
(continues on next page)

(continued from previous page)

```
For a person in the privileged group, the model predict a correct positive output 4.90
↪% more often than a person in the unprivileged group. This is considered to be fair.
For a person in the privileged group, the model predict a correct positive output or␣
↪a correct negative output 2.62% more often than a person in the unprivileged group.␣
↪This is considered to be fair.
The model has 3 fair metrics over 4 (75%).
```

And if you like to get visual help use the *plot_bias* function :

```
>>> privileged_group = {'gender': ['Male']}
>>> from transparentai import fairness
>>> fairness.plot_bias(y_true_valid, y_pred_valid, df_valid, privileged_group, with_
↪text=True)
```

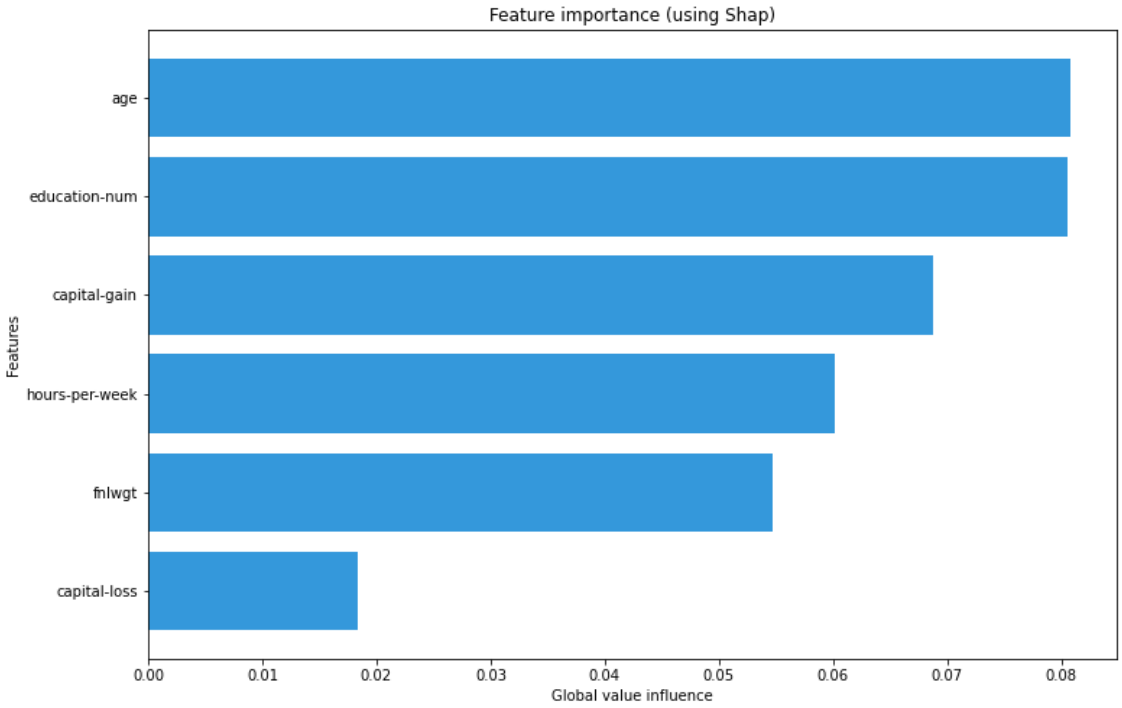


1.2.2 How can I explain my model ?

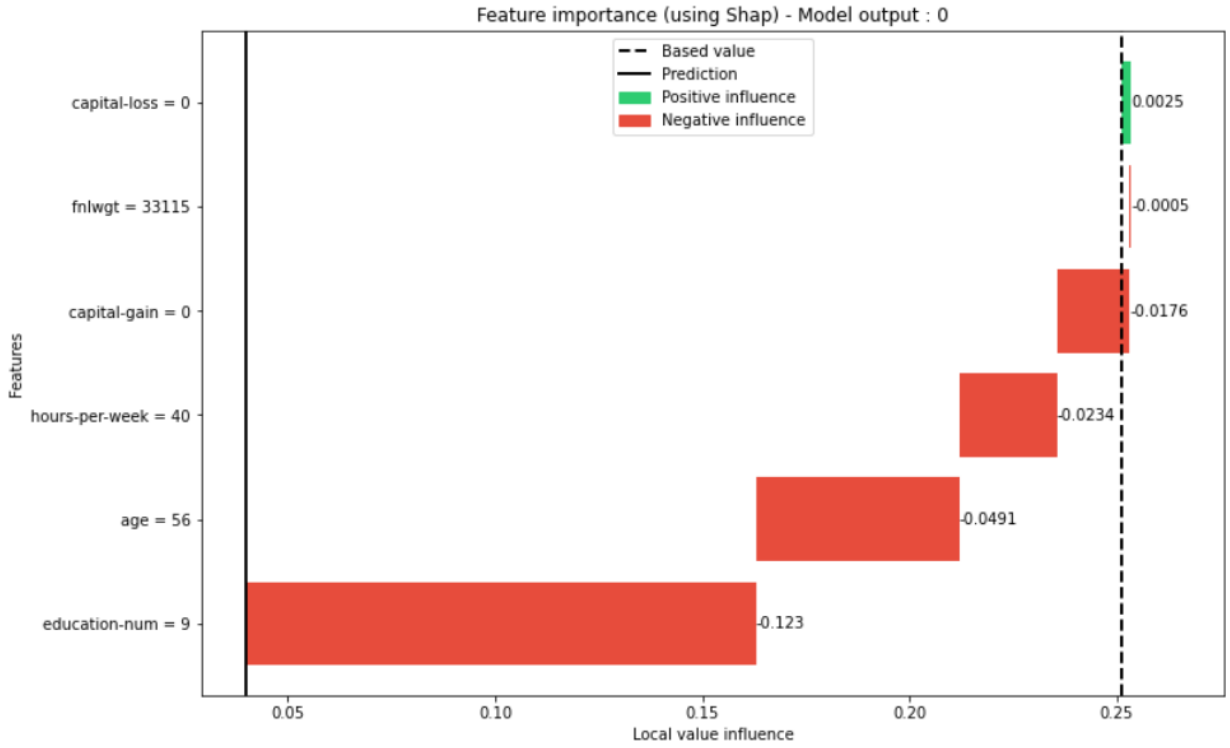
```
>>> from transparentai.models import explainers
>>> explainer = explainers.ModelExplainer(clf, X_train, model_type='tree')
```

```
>>> explainer.explain_global_influence(X_train, nsamples=1000)
{
    'age': 0.08075649984055841,
    'fnlwgt': 0.05476459574744569,
    'education-num': 0.08048316800088552,
    'capital-gain': 0.06879137962639843,
    'capital-loss': 0.018367250661071737,
    'hours-per-week': 0.06009733425389803
}
```

```
>>> explainer.plot_global_explain()
```




```
>>> explainer.plot_local_explain(X_valid.iloc[0])
```



1.2.3 What's my model performance ?

```
>>> from transparentai.models import classification
```

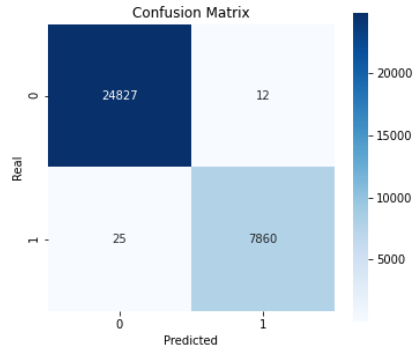
```
>>> # You can use custom function with lambda
>>> metrics = ['accuracy', 'roc_auc', 'f1', 'recall', 'precision', lambda y_true, y_
↪pred: sum(y_true-y_pred)]
>>> classification.compute_metrics(y_true_valid, y_pred_valid, metrics)
{
    'accuracy': 0.812011415808413,
    'roc_auc': 0.8272860034692258,
    'f1': 0.5682530635508691,
    'recall': 0.5244608100999474,
    'precision': 0.6200248756218906,
    'custom_1': 586
}
```

```
>>> classification.plot_performance(y_true, y_pred, y_true_valid, y_pred_valid)
```

Model performance plot train set (left) and test set (right)

Table of scores

accuracy	0.9989
f1	0.9977
recall	0.9968
precision	0.9985



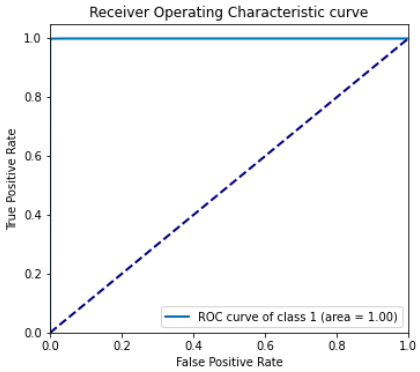
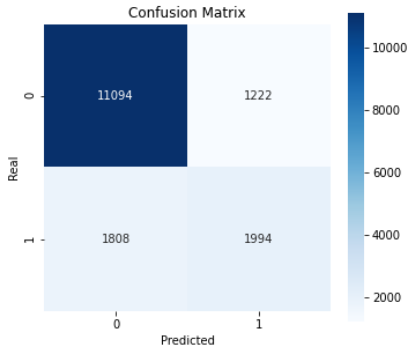
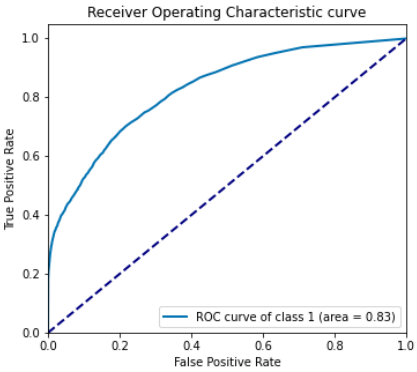


Table of scores

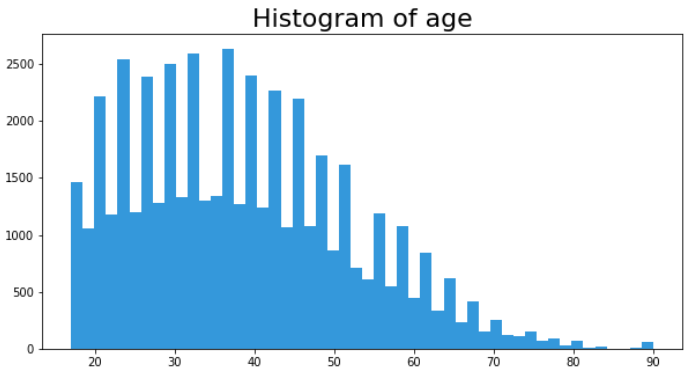
accuracy	0.812
f1	0.5683
recall	0.5245
precision	0.62





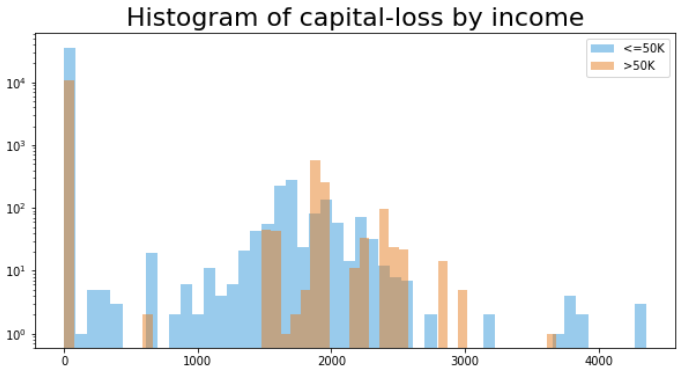
1.2.4 What is in my data ?

```
>>> from transparentai.datasets import variable
>>> variable.plot_variable(data['age'])
```



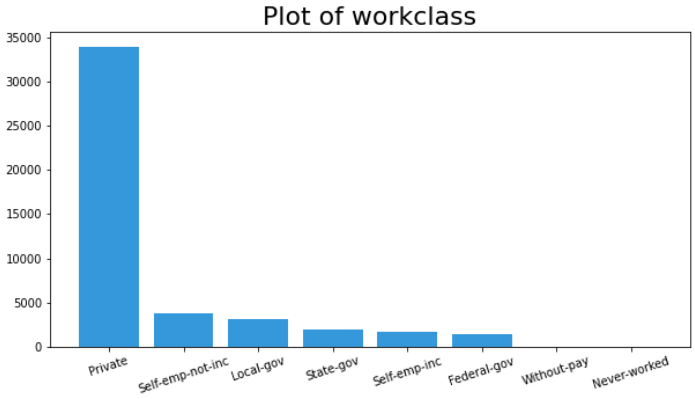
Valid Values	48842 (100%)
Missing Values	0 (0%)
Mean	38.6436
Mode	36
Min	17
Quantile 25%	28.0
Quantile 50%	37.0
Quantile 75%	48.0
Max	90

```
>>> variable.plot_variable(data['capital-loss'], legend=data['income'], ylog=True)
```



Valid Values	48842 (100%)
Missing Values	0 (0%)
Mean	87.5023
Mode	0
Min	0
Quantile 25%	0.0
Quantile 50%	0.0
Quantile 75%	0.0
Max	4356

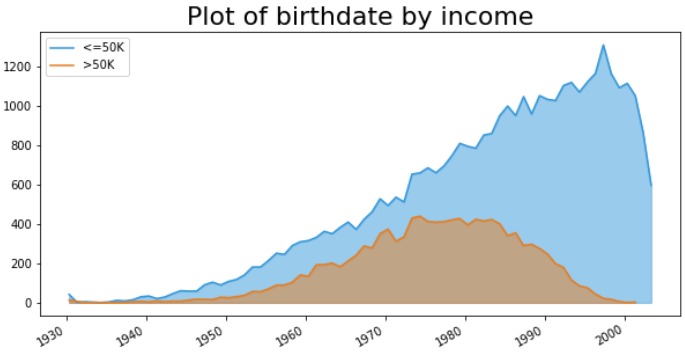
```
>>> variable.plot_variable(data['workclass'])
```



Valid Values	46043 (94%)
Missing Values	2799 (5%)
Unique Values	8
Most Common	Private

The *birthdate* column was generated based on the *age* column.

```
>>> variable.plot_variable(data['birthdate'], legend=data['income'])
```

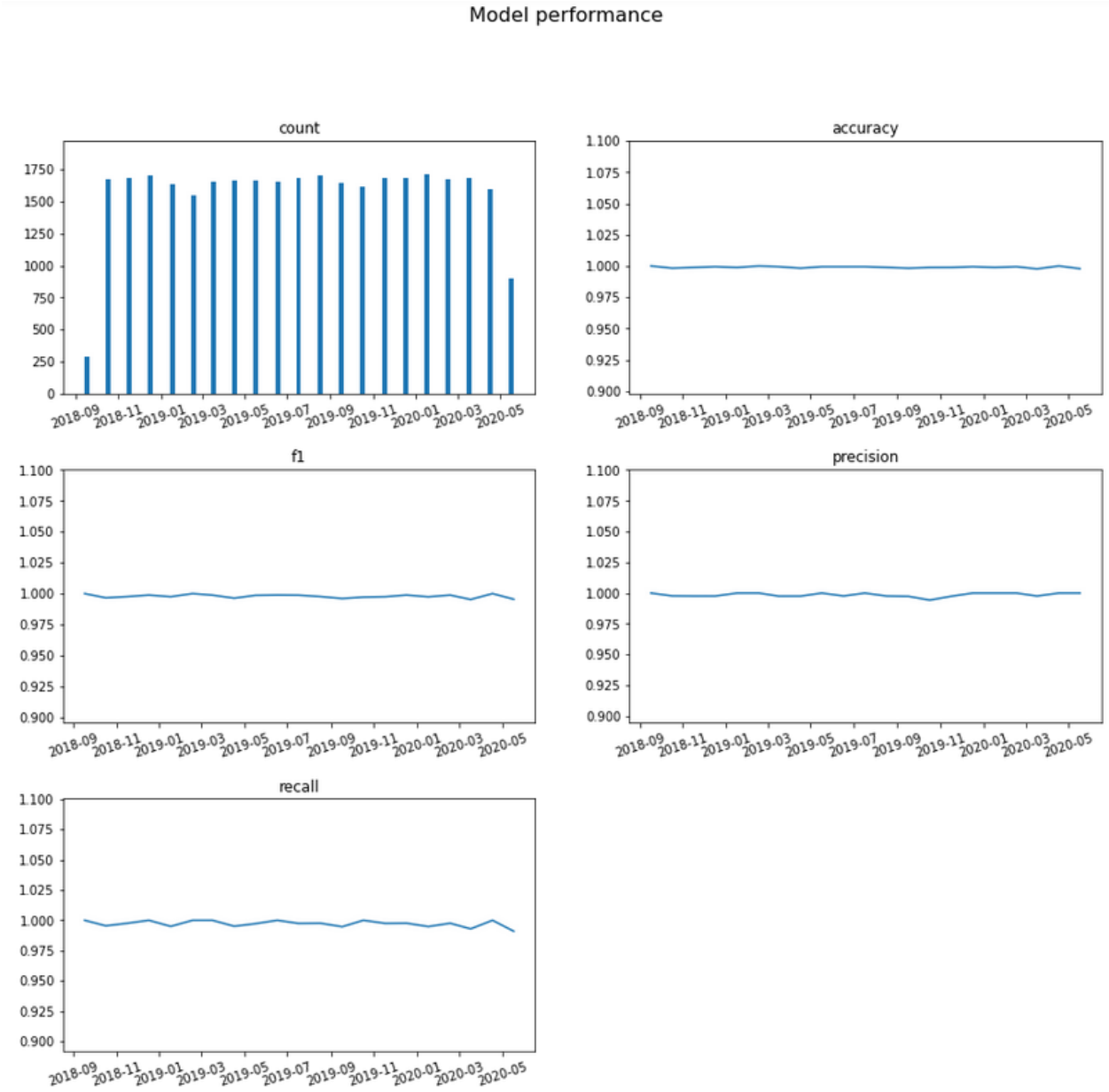


Valid Values	48842 (100%)
Missing Values	0 (0%)
Unique Values	16417
Most Common	1984-05-28
Min	1930-01-04
Mean	1981-11-07
Max	2003-12-28

1.2.5 How can I know the model is still good over time ?

timestamp variable was generated randomly, it represents the time of the prediction.

```
>>> from transparentai import monitoring
>>> monitoring.plot_monitoring(y_true, y_pred, timestamp, interval='month',␣
↪classification=True)
```



1.2.6 Is my model sustainable ?

Estimate your training CO2 consumption.

```
>>> from transparentai import sustainable
>>> sustainable.estimate_co2(hours=24, location='France', watts=250)
3.18437946896484
```

Evaluate your training kWh consumption.

```
>>> from transparentai import sustainable
>>> kWh, clf = sustainable.evaluate_kWh(clf.fit, X, Y, verbose=True)
Location:                                                                  France
Baseline wattage:                                                        4.79 watts
Process wattage:                                                        18.45 watts
--------------------------------------------------------------------------------
------------------------------  Final Readings  -------------------------------
--------------------------------------------------------------------------------
Average baseline wattage:                                                3.53 watts
Average total wattage:                                                  16.04 watts
Average process wattage:                                                12.51 watts
Process duration:                                                          0:00:07
--------------------------------------------------------------------------------
------------------------------   Energy Data    -------------------------------
--------------------------------------------------------------------------------
                              Energy mix in France
Coal:                                                                      3.12%
Petroleum:                                                                16.06%
Natural Gas:                                                              33.56%
Low Carbon:                                                               47.26%
--------------------------------------------------------------------------------
------------------------------    Emissions     -------------------------------
--------------------------------------------------------------------------------
Effective emission:                                              1.32e-05 kg CO2
Equivalent miles driven:                                          5.39e-12 miles
Equivalent minutes of 32-inch LCD TV watched:                    8.14e-03 minutes
Percentage of CO2 used in a US household/day:                          4.33e-12%
--------------------------------------------------------------------------------
------------------------- Assumed Carbon Equivalencies -------------------------
--------------------------------------------------------------------------------
Coal:                                                        995.725971 kg CO2/MWh
Petroleum:                                                  816.6885263 kg CO2/MWh
Natural gas:                                                743.8415916 kg CO2/MWh
Low carbon:                                                           0 kg CO2/MWh
--------------------------------------------------------------------------------
-------------------------     Emissions Comparison     -------------------------
--------------------------------------------------------------------------------
                     Quantities below expressed in kg CO2
        US                        Europe                    Global minus US/Europe
Max:    Wyoming        2.85e-05 Kosovo         2.93e-05 Mongolia        2.86e-05
Median: Tennessee      1.40e-05 Ukraine        2.04e-05 Korea, South    2.34e-05
Min:    Vermont        8.00e-07 Iceland        5.26e-06 Bhutan          3.26e-06
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Process used:                                                    3.10e-05 kWh
```

1.2.7 Do I use safe packages ?

```
>>> import transparentai.utils as utils
>>> utils.check_packages_security(full_report=True)
+=================================================================================+
|                                                                                 |
|                                /$$$$$$            /$$                           |
|                               /$$__  $$          | $$                           |
|           /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$           |
```

(continues on next page)

(continued from previous page)

```
|          /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$           |
|         |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$           |
|          \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$           |
|          /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$           |
|         |_______/  \_______/|__/     \_______/   \___/   \____  $$           |
|                                                              /$$  | $$           |
|                                                             |  $$$$$$/           |
|  by pyup.io                                                  \______/            |
|                                                                                  |
+==================================================================================+
| REPORT                                                                           |
| checked 77 packages, using default DB                                            |
+==================================================================================+
| No known security vulnerabilities found.                                         |
+==================================================================================+
```

1.3 List of preformated metrics

1.3.1 Evaluation metrics

If you use the compute_metrics function in the `classification.compute_metrics` or `regression.compute_metrics` functions there are preformated metrics.

You can see details of the code in the documentation page : transparentai.models

This is the list :

Problem type	metric name
classification	`'accuracy'`
classification	`'balanced_accuracy'`
classification	`'average_precision'`
classification	`'brier_score'`
classification	`'f1'`
classification	`'f1_micro'`
classification	`'f1_macro'`
classification	`'f1_weighted'`
classification	`'f1_samples'`
classification	`'log_loss'`
classification	`'precision'`
classification	`'precision_micro'`
classification	`'recall'`
classification	`'recall_micro'`
classification	`'true_positive_rate'`
classification	`'false_positive_rate'`
classification	`'jaccard'`
classification	`'matthews_corrcoef'`
classification	`'roc_auc'`
classification	`'roc_auc_ovr'`
classification	`'roc_auc_ovo'`
classification	`'roc_auc_ovr_weighted'`

Continued on next page

Table 1 – continued from previous page

Problem type	metric name
classification	`'roc_auc_ovo_weighted'`
classification	`'true_positives'`
classification	`'false_positives'`
classification	`'false_negatives'`
classification	`'true_negatives'`
classification	`'confusion_matrix'`
regression	`'max_error'`
regression	`'mean_absolute_error'`
regression	`'mean_squared_error'`
regression	`'root_mean_squared_error'`
regression	`'mean_squared_log_error'`
regression	`'median_absolute_error'`
regression	`'r2'`
regression	`'mean_poisson_deviance'`
regression	`'mean_gamma_deviance'`

1.4 transparentai.datasets

1.4.1 Variable submodule

transparentai.datasets.variable.variable.**describe_number**(*arr*)

Descriptive statistics about a number array.

Returned statistics:

- Count of valid values
- Count of missing values
- Mean
- Mode
- Min
- Quantitle 25%
- Median
- Quantile 75%
- Max

Parameters **arr** (*array like*) – Array of value to get desriptive statistics from

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not a number array

transparentai.datasets.variable.variable.**describe_datetime**(*arr*, *format='%Y-%m-%d'*)

Descriptive statistics about a datetime array.

Returned statistics:

- Count of valid values
- Count of missing values
- Count of unique values
- Most common value
- Min
- Mean
- Max

Parameters

- **arr** (*array like*) – Array of value to get desriptive statistics from
- **format** (*str*) – String format for datetime value

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not a datetime array

transparentai.datasets.variable.variable.**describe_object**(*arr*)

Descriptive statistics about an object array.

Returned statistics:

- Count of valid values
- Count of missing values
- Count of unique values
- Most common value

Parameters **arr** (*array like*) – Array of value to get desriptive statistics from

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not an object array

transparentai.datasets.variable.variable.**describe**(*arr*)

Descriptive statistics about an array. Depending on the detected dtype (number, date, object) it returns specific stats.

Common statistics for all dtype (using describe_common):

- Count of valid values
- Count of missing values

Number statistics (using describe_number):

- Mean
- Mode
- Min
- Quantitle 25%
- Median

- Quantile 75%
- Max

Datetime statistics (using describe_datetime):

- Count of unique values
- Most common value
- Min
- Mean
- Max

Object statistics (using describe_datetime):

- Count of unique values
- Most common value

Parameters **arr** (*array like*) – Array of value to get desriptive statistics from

Returns Dictionnary with descriptive statistics

Return type dict

Raises TypeError: – arr is not an array like

transparentai.datasets.variable.correlation.**compute_correlation**(*df*, *nrows=None*, *max_cat_val=100*)

Computes differents correlations matrix for three cases and merge them:

- numerical to numerical (using Pearson coeff)
- categorical to categorical (using Cramers V & Chi square)
- numerical to categorical (discrete) (using Point Biserial)

This matrix has a default : the cramers_v_corr is scale from 0 to 1, but the others are from to -1 to 1. Be sure to understand this.

Pearson coeff Wikipedia definition :

In statistics, the Pearson correlation coefficient, also referred to as Pearson's r, the Pearson product-moment correlation coefficient (PPMCC) or the bivariate correlation, is a statistic that measures linear correlation between two variables X and Y. It has a value between +1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and 1 is total negative linear correlation (that the value lies between -1 and 1 is a consequence of the Cauchy–Schwarz inequality). It is widely used in the sciences.

Cramers V Wikipedia definition :

In statistics, Cramér's V (sometimes referred to as Cramér's phi and denoted as ϕc) is a measure of association between two nominal variables, giving a value between 0 and +1 (inclusive). It is based on Pearson's chi-squared statistic and was published by Harald Cramér in 1946.

Point Biserial Wikipedia definition :

The point biserial correlation coefficient (rpb) is a correlation coefficient used when one variable (e.g. Y) is dichotomous; Y can either be "naturally" dichotomous, like whether a coin lands heads or tails, or an artificially dichotomized variable. In most situations it is not advisable to dichotomize variables artificially[citation needed]. When a new variable is artificially dichotomized the new dichotomous variable may be conceptualized as having an underlying continuity. If this is the case, a biserial correlation would be the more appropriate calculation.

Parameters

- **df** (*pd.DataFrame*) – pandas Dataframe with values to compute correlation
- **nrows** (*None or int or float (default None)*) – If not None reduce the data to a sample of nrows if int else if float reduce to len(df) * nrows
- **max_cat_val** (*int or None (default 100)*) – Number max of unique values in a categorical feature if there are more distinct values than this number then the feature is ignored

Returns Correlation matrix computed with Pearson coeff for numerical features to numerical features, Cramers V for categorical features to categorical features and Point Biserial for categorical features to numerical features

Return type pd.DataFrame

Raises TypeError: – Must provide a pandas DataFrame representing the data

transparentai.datasets.variable.correlation.**compute_cramers_v_corr**(*df*)

Computes Cramers V correlation for a dataframe.

Cramers V Wikipedia definition :

In statistics, Cramér's V (sometimes referred to as Cramér's phi and denoted as ϕc) is a measure of association between two nominal variables, giving a value between 0 and +1 (inclusive). It is based on Pearson's chi-squared statistic and was published by Harald Cramér in 1946.

Parameters **df** (*pd.DataFrame*) – pandas Dataframe with values to compute Cramers V correlation

Returns Correlation matrix computed for Cramers V coeff

Return type pd.DataFrame

Raises TypeError: – Must provide a pandas DataFrame representing the data

transparentai.datasets.variable.correlation.**compute_pointbiserialr_corr**(*df*, *cat_feats=None*, *num_feats=None*)

Computes Point Biserial correlation for a dataframe.

Point Biserial Wikipedia definition :

The point biserial correlation coefficient (rpb) is a correlation coefficient used when one variable (e.g. Y) is dichotomous; Y can either be "naturally" dichotomous, like whether a coin lands heads or tails, or an artificially dichotomized variable. In most situations it is not advisable to dichotomize variables artificially[citation needed]. When a new variable is artificially dichotomized the new dichotomous variable may be conceptualized as having an underlying continuity. If this is the case, a biserial correlation would be the more appropriate calculation.

Parameters **df** (*pd.DataFrame*) – pandas Dataframe with values to compute Point Biserial correlation

Returns Correlation matrix computed for Point Biserial coeff

Return type pd.DataFrame

Raises

- TypeError: – Must provide a pandas DataFrame representing the data
- ValueError: – cat_feats and num_feats must be set or be both None
- TypeError: – cat_feats must be a list

- TypeError: – num_feats must be a list

transparentai.datasets.variable.correlation.**cramers_v**(*x*, *y*)
Returns the Cramer V value of two categorical variables using chi square. This correlation metric is between 0 and 1.

Code source found in this article : https://towardsdatascience.com/the-search-for-categorical-correlation-a1cf7f1888c9

Parameters

- **x** (*array like*) – first categorical variable
- **y** (*array like*) – second categorical variable

Returns Cramer V value

Return type float

transparentai.datasets.variable.correlation.**merge_corr_df**(*df_list*)
Merges correlation matrix from compute_correlation() function to one. Needs 3 dataframe : pearson_corr, cramers_v_corr and pbs_corr.

This matrix has a default : the cramers_v_corr is scale from 0 to 1, but the others are from to -1 to 1. Be sure to understand this.

Parameters **df_list** (*list*) – List of correlation matrices

Returns Merged dataframe of correlation matrices

Return type pd.DataFrame

1.4.2 Preformated datasets

transparentai.datasets.datasets.**load_adult**()
Load Adult dataset. Source : https://archive.ics.uci.edu/ml/datasets/Adult

transparentai.datasets.datasets.**load_boston**()
Load boston dataset Source : https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

transparentai.datasets.datasets.**load_iris**()
Load Iris dataset. Source : http://archive.ics.uci.edu/ml/datasets/Iris/

1.5 transparentai.fairness

1.5.1 Fairness module

transparentai.fairness.fairness.**create_privilieged_df**(*df*, *privileged_group*)
Returns a formated dataframe with protected attribute columns and whether the row is privileged (1) or not (0).

example of a privileged_group dictionnary :

```
>>> privileged_group = {
     'gender':['Male'],                # privileged group is man for gender␣
↪attribute
     'age': lambda x: x > 30 & x < 55  # privileged group aged between 30 and 55␣
↪years old
 }
```

Parameters

- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group

Returns DataFrame with protected attribute columns and whether the row is privileged (1) or not (0)

Return type pd.DataFrame

Raises

- TypeError: – df is not a pandas.DataFrame
- TypeError: – privileged_group is not a dictionnary
- ValueError: – privileged_group has not valid keys (in df columns)

transparentai.fairness.fairness.**compute_fairness_metrics**(*y_true*, *y_pred*, *df*, *privileged_group*, *metrics=None*, *pos_label=1*, *regr_split=None*)

Computes the fairness metrics for one attribute

metrics can have str or function. If it's a string then it has to be a key from FAIRNESS_METRICS global variable dict. By default it uses the 5 fairness function :

- statistical_parity_difference
- disparate_impact
- equal_opportunity_difference
- average_odds_difference
- theil_index

You can also use it for a regression problem. You can set a value in the regr_split argument so it converts it to a binary classification problem. To use the mean use 'mean'. If the favorable label is more than the split value set pos_label argument to 1 else to 0.

Example

```
>>> from transparentai.datasets import load_boston
>>> from sklearn.linear_model import LinearRegression
```

```
>>> data = load_boston()
>>> X, y = data.drop(columns='MEDV'), data['MEDV']
>>> regr = LinearRegression().fit(X, y)
```

```
>>> privileged_group = {
    'AGE': lambda x: (x > 30) & (x < 55)
}
>>> y_true, y_pred = y, regr.predict(X)
>>> compute_fairness_metrics(y_true, y_pred, data,
                             privileged_group, regr_split='mean')
{'AGE': {'statistical_parity_difference': -0.2041836536594836,
```

(continues on next page)

(continued from previous page)

```
  'disparate_impact': 0.674582301980198,
  'equal_opportunity_difference': 0.018181818181818188,
  'average_odds_difference': -0.0884835589941973,
  'theil_index': 0.06976073748626294}}
```

Returns a dictionnary with protected attributes name's as key containing a dictionnary with metric's name as key and metric function's result as value

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group
- **metrics** (*list (default None)*) – List of metrics to compute, if None then it uses the 5 default Fairness function
- **pos_label** (*number*) – The label of the positive class.
- **regr_split** (*'mean' or number (default None)*) – If it's a regression problem then you can convert result to a binary classification using 'mean' or a choosen number. both y_true and y_pred become 0 and 1 : 0 if it's equal or less than the split value (the average if 'mean') and 1 if more. If the favorable label is more than the split value set pos_label=1 else pos_label=0

Returns Dictionnary with protected attributes name's as key containing a dictionnary with metric's name as key and metric function's result as value

Return type dict

Raises

- ValueError: – y_true and y_pred must have the same length
- ValueError: – y_true and df must have the same length
- TypeError: – metrics must be a list

transparentai.fairness.fairness.**model_bias**(*y_true*, *y_pred*, *df*, *privileged_group*, *pos_label=1*, *regr_split=None*, *returns_text=False*)

Computes the fairness metrics for protected attributes refered in the privileged_group argument.

It uses the 4 fairness function :

- statistical_parity_difference
- disparate_impact
- equal_opportunity_difference
- average_odds_difference

You can also use it for a regression problem. You can set a value in the regr_split argument so it converts it to a binary classification problem. To use the mean use 'mean'. If the favorable label is more than the split value set pos_label argument to 1 else to 0.

This function is using the fairness.compute_metrics function. So if returns_text is False then it's the same output.

Example

```
>>> from transparentai.datasets import load_boston
>>> from sklearn.linear_model import LinearRegression
```

```
>>> data = load_boston()
>>> X, y = data.drop(columns='MEDV'), data['MEDV']
>>> regr = LinearRegression().fit(X, y)
```

```
>>> privileged_group = {
    'AGE': lambda x: (x > 30) & (x < 55)
}
>>> y_true, y_pred = y, regr.predict(X)
>>> model_bias(y_true, y_pred, data,
               privileged_group, regr_split='mean')
{'AGE': {'statistical_parity_difference': -0.2041836536594836,
  'disparate_impact': 0.674582301980198,
  'equal_opportunity_difference': 0.018181818181818188,
  'average_odds_difference': -0.0884835589941973,
  'theil_index': 0.06976073748626294}}
```

```
>>> bias_txt = model_bias(y_true, y_pred, data,
                          privileged_group, regr_split='mean',
                          returns_text=True)
>>> print(bias_txt['AGE'])
The privileged group is predicted with the positive output 20.42% more often than␣
↪the unprivileged group. This is considered to be not fair.
The privileged group is predicted with the positive output 1.48 times more often␣
↪than the unprivileged group. This is considered to be not fair.
For a person in the unprivileged group, the model predict a correct positive␣
↪output 1.82% more often than a person in the privileged group. This is␣
↪considered to be fair.
For a person in the privileged group, the model predict a correct positive output␣
↪or a correct negative output 8.85% more often than a person in the unprivileged␣
↪group. This is considered to be fair.
The model has 2 fair metrics over 4 (50%).
```

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group
- **pos_label** (*number*) – The label of the positive class.
- **regr_split** (*'mean' or number (default None)*) – If it's a regression problem then you can convert result to a binary classification using 'mean' or a choosen number. both y_true and y_pred become 0 and 1 : 0 if it's equal or less than the split value (the average if 'mean') and 1 if more. If the favorable label is more than the split value set pos_label=1 else pos_label=0

- **returns_text** (*bool (default False)*) – Whether it return computed metrics score or a text explaination for the computed bias.

Returns Dictionnary with metric's name as key and metric function's result as value if returns_text is False else it returns a text explaining the model fairness over the 4 metrics.

Return type dict

transparentai.fairness.fairness.**find_correlated_feature**(*df*, *privileged_group*, *corr_threshold=0.4*)

Finds correlated feature with protected attribute set in the privileged_group argument.

This function is a helper to find out if protected attribute can be found in other features.

Returns a dictionnary with protected attributes name's as key containing a dictionnary with metric's name as key and metric function's result as value.

Example

```
>>> from transparentai.datasets import load_adult
>>> from transparentai import fairness
```

```
>>> data = load_adult()
```

```
>>> privileged_group = {
        'gender':['Male'],
        'marital-status': lambda x: 'Married' in x,
        'race':['White']
    }
```

```
>>> fairness.find_correlated_feature(data, privileged_group,
                                     corr_threshold=0.4)
{'gender': {'marital-status': 0.4593,
  'occupation': 0.4239,
  'relationship': 0.6465},
 'marital-status': {'relationship': 0.4881,
  'gender': 0.4593,
  'income': 0.4482},
 'race': {'native-country': 0.4006}}
```

Parameters

- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group
- **corr_threshold** (*float (default 0.4)*) – Threshold for which features are considered to be correlated

Returns Dictionnary with protected attributes name's as key containing a dictionnary with correlated features as key and correlation coeff as value

Return type dict

1.5.2 Fairness metrics

Statistical parity difference

transparentai.fairness.metrics.**statistical_parity_difference**(*y*, *prot_attr*, *pos_label=1*)

Computes the statistical parity difference for a protected attribute and a specified label

Computed as the difference of the rate of favorable outcomes received by the unprivileged group to the privileged group.

The ideal value of this metric is 0 A value < 0 implies higher benefit for the privileged group and a value > 0 implies a higher benefit for the unprivileged group.

Fairness for this metric is between -0.1 and 0.1

$$Pr(\hat{Y} = v|D = \text{unprivileged}) - Pr(\hat{Y} = v|D = \text{privileged})$$

code source inspired from aif360 statistical_parity_difference

Parameters

- **y** (*array like*) – list of predicted labels
- **prot_attr** (*array like*) – Array of 0 and 1 same length as y which indicates if the row is member of a privileged group or not
- **pos_label** (*int (default 1)*) – number of the positive label

Returns Statistical parity difference bias metric

Return type float

Raises ValueError: – y and prot_attr must have the same length

Disparate Impact

transparentai.fairness.metrics.**disparate_impact**(*y*, *prot_attr*, *pos_label=1*)

Computes the Disparate impact for a protected attribute and a specified label

Computed as the ratio of rate of favorable outcome for the unprivileged group to that of the privileged group.

The ideal value of this metric is 1.0 A value < 1 implies higher benefit for the privileged group and a value > 1 implies a higher benefit for the unprivileged group.

Fairness for this metric is between 0.8 and 1.2

$$\frac{Pr(\hat{Y} = v|D = \text{unprivileged})}{Pr(\hat{Y} = v|D = \text{privileged})}$$

code source inspired from aif360 disparate_impact

Parameters

- **y** (*array like*) – list of predicted labels
- **prot_attr** (*array like*) – Array of 0 and 1 same length as y which indicates if the row is member of a privileged group or not
- **pos_label** (*int (default 1)*) – number of the positive label

Returns Disparate impact bias metric

Return type float

Raises ValueError: – y and prot_attr must have the same length

Equal opportunity difference

transparentai.fairness.metrics.**equal_opportunity_difference**(*y_true*, *y_pred*, *prot_attr*, *pos_label=1*)

Computes the equal opportunity difference for a protected attribute and a specified label

This metric is computed as the difference of true positive rates between the unprivileged and the privileged groups. The true positive rate is the ratio of true positives to the total number of actual positives for a given group.

The ideal value is 0. A value of < 0 implies higher benefit for the privileged group and a value > 0 implies higher benefit for the unprivileged group.

Fairness for this metric is between -0.1 and 0.1

$TPR_{D=\text{unprivileged}} - TPR_{D=\text{privileged}}$

code source from aif360 equal_opportunity_difference

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **prot_attr** (*array like*) – Array of 0 and 1 same length as y which indicates if the row is member of a privileged group or not
- **pos_label** (*int (default 1)*) – number of the positive label

Returns Equal opportunity difference bias metric

Return type float

Raises

- ValueError: – y_true and y_pred must have the same length
- ValueError: – y_true and prot_attr must have the same length

Average odds difference

transparentai.fairness.metrics.**average_odds_difference**(*y_true*, *y_pred*, *prot_attr*, *pos_label=1*)

Computes the average odds difference for a protected attribute and a specified label

Computed as average difference of false positive rate (false positives / negatives) and true positive rate (true positives / positives) between unprivileged and privileged groups.

The ideal value of this metric is 0. A value of < 0 implies higher benefit for the privileged group and a value > 0 implies higher benefit for the unprivileged group.

Fairness for this metric is between -0.1 and 0.1

$$\frac{1}{2}\left[|FPR_{D=\text{unprivileged}} - FPR_{D=\text{privileged}}| + |TPR_{D=\text{unprivileged}} - TPR_{D=\text{privileged}}|\right]$$

A value of 0 indicates equality of odds.

code source from aif360 average_odds_difference

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **prot_attr** (*array like*) – Array of 0 and 1 same length as y which indicates if the row is member of a privileged group or not
- **pos_label** (*int (default 1)*) – number of the positive label

Returns Average of absolute difference bias metric

Return type float

Raises

- ValueError: – y_true and y_pred must have the same length
- ValueError: – y_true and prot_attr must have the same length

Theil Index

transparentai.fairness.metrics.**theil_index**(*y_true*, *y_pred*, *prot_attr*, *pos_label=1*)

Computes the theil index for a protected attribute and a specified label

Computed as the generalized entropy of benefit for all individuals in the dataset, with alpha = 1. It measures the inequality in benefit allocation for individuals.

A value of 0 implies perfect fairness.

Fairness is indicated by lower scores, higher scores are problematic

With $b_i = \hat{y}_i - y_i + 1$:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{b_i}{\mu}\ln\frac{b_i}{\mu}$$

code source from aif360 theil_index

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **prot_attr** (*array like*) – Array of 0 and 1 same length as y which indicates if the row is member of a privileged group or not
- **pos_label** (*int (default 1)*) – number of the positive label

Returns Theil index bias metric

Return type float

Raises

- ValueError: – y_true and y_pred must have the same length
- ValueError: – y_true and prot_attr must have the same length

1.6 transparentai.models

1.6.1 Evaluation submodule

transparentai.models.evaluation.evaluation.**compute_metrics**(*y_true*, *y_pred*, *metrics*, *classification=True*)

Computes the inputed metrics.

metrics can have str or function. If it's a string then it has to be a key from METRICS global variable dict.

Returns a dictionnary with metric's name as key and metric function's result as value

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **metrics** (*list*) – List of metrics to compute
- **classification** (*bool (default True)*) – Whether the ML task is a classification or not

Returns Dictionnary with metric's name as key and metric function's result as value

Return type dict

Raises TypeError: – metrics must be a list

1.6.2 Classification metrics

transparentai.models.evaluation.classification.**accuracy**(*y_true*, *y_pred*, ***args*)

Accuracy score based on the sklearn.metrics.accuracy_score function.

More details here : Accuracy score

transparentai.models.evaluation.classification.**average_precision**(*y_true*, *y_prob*, ***args*)

Average prevision score based on the sklearn.metrics.average_precision_score function.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**balanced_accuracy**(*y_true*, *y_pred*, ***args*)

Balanced accuracy score based on the sklearn.metrics.balanced_accuracy_score function.

More details here : Balanced accuracy score

transparentai.models.evaluation.classification.**brier_score**(*y_true*, *y_prob*, ***args*)

Brier score based on the sklearn.metrics.brier_score_loss function.

More details here : Probability calibration

transparentai.models.evaluation.classification.**confusion_matrix**(*y_true*, *y_pred*, ***args*)

Confusion matrix based on the sklearn.metrics.confusion_matrix function.

More details here : Confusion matrix

transparentai.models.evaluation.classification.**f1**(*y_true*, *y_pred*, ***args*)
F1 score based on the sklearn.metrics.f1_score function.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**f1_macro**(*y_true*, *y_pred*, ***args*)
F1 score based on the sklearn.metrics.f1_score function.

Average argument set to 'macro'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**f1_micro**(*y_true*, *y_pred*, ***args*)
F1 score based on the sklearn.metrics.f1_score function.

Average argument set to 'micro'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**f1_samples**(*y_true*, *y_pred*, ***args*)
F1 score based on the sklearn.metrics.f1_score function.

Average argument set to 'samples'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**f1_weighted**(*y_true*, *y_pred*, ***args*)
F1 score based on the sklearn.metrics.f1_score function.

Average argument set to 'weighted'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**false_negatives**(*y_true*, *y_pred*, *pos_label=1*)
Returns the number of false negatives given a class number.

$$FN = \sum_{i}^{n} (y_i = 1) \& (\hat{y}_i \neq 1)$$

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **pos_label** (*int (default 1)*) – Label class number (if binary classification then it's 1)

Returns Number of false negatives

Return type int

transparentai.models.evaluation.classification.**false_positive_rate**(*y_true*, *y_pred*, *pos_label=1*)

transparentai.models.evaluation.classification.**false_positives**(*y_true*, *y_pred*, *pos_label=1*)
Returns the number of false positives given a class number.

$$FP = \sum_{i}^{n} (y_i \neq 1) \& (\hat{y}_i = 1)$$

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **pos_label** (*int (default 1)*) – Label class number (if binary classification then it's 1)

Returns Number of false positives

Return type int

transparentai.models.evaluation.classification.**jaccard**(*y_true*, *y_pred*, ***args*)
Jaccard score based on the sklearn.metrics.jaccard_score function.

More details here : Jaccard similarity coefficient score

transparentai.models.evaluation.classification.**log_loss**(*y_true*, *y_prob*, ***args*)
Log loss based on the sklearn.metrics.log_loss function.

More details here : Log loss

transparentai.models.evaluation.classification.**matthews_corrcoef**(*y_true*, *y_pred*, ***args*)
Matthews correlation coefficient based on the sklearn.metrics.matthews_corrcoef function.

More details here : Matthews correlation coefficient

transparentai.models.evaluation.classification.**precision**(*y_true*, *y_pred*, ***args*)
Precision score based on the sklearn.metrics.precision_score function.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**precision_micro**(*y_true*, *y_pred*, ***args*)
Precision score based on the sklearn.metrics.precision_score function.

Average argument set to 'micro'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**recall**(*y_true*, *y_pred*, ***args*)
Recall score based on the sklearn.metrics.recall_score function.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**recall_micro**(*y_true*, *y_pred*, ***args*)
Recall score based on the sklearn.metrics.recall_score function.

Average argument set to 'micro'.

More details here : Precision, recall and F-measures

transparentai.models.evaluation.classification.**roc_auc**(*y_true*, *y_prob*, ***args*)
Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**roc_auc_ovo**(*y_true*, *y_prob*, ***args*)
Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

multi_class argument is set to 'ovo'.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**roc_auc_ovo_weighted**(*y_true*, *y_prob*, ***args*)

Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

Average argument set to 'weighted' and multi_class to 'ovo'.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**roc_auc_ovr**(*y_true*, *y_prob*, ***args*)

Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

multi_class argument is set to 'ovr'.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**roc_auc_ovr_weighted**(*y_true*, *y_prob*, ***args*)

Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

Average argument set to 'weighted' and multi_class to 'ovr'.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**roc_curve**(*y_true*, *y_prob*, ***args*)

Area Under the Receiver Operating Characteristic Curve (ROC AUC) score based on the sklearn.metrics.roc_auc_score function.

More details here : Receiver operating characteristic (ROC)

transparentai.models.evaluation.classification.**true_negatives**(*y_true*, *y_pred*, *pos_label=1*)

Returns the number of true negatives given a class number.

$$TN = \sum_{i}^{n} (y_i \neq 1) \& (\hat{y}_i \neq 1)$$

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **pos_label** (*int (default 1)*) – Label class number (if binary classification then it's 1)

Returns Number of true negatives

Return type int

transparentai.models.evaluation.classification.**true_positive_rate**(*y_true*, *y_pred*, *pos_label=1*)

transparentai.models.evaluation.classification.**true_positives**(*y_true*, *y_pred*, *pos_label=1*)

Returns the number of true positives given a class number.

$$TP = \sum_{i}^{n} (y_i = 1) \& (\hat{y}_i = 1)$$

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **pos_label** (*int (default 1)*) – Label class number (if binary classification then it's 1)

Returns Number of true positives

Return type int

1.6.3 Regression metrics

transparentai.models.evaluation.regression.**explained_variance**(*y_true*, *y_pred*, ***args*)

Explained variance score based on the sklearn.metrics.explained_variance_score function.

More details here : Explained variance score

transparentai.models.evaluation.regression.**max_error**(*y_true*, *y_pred*, ***args*)

Max error based on the sklearn.metrics.max_error function.

More details here : Max error

transparentai.models.evaluation.regression.**mean_absolute_error**(*y_true*, *y_pred*, ***args*)

Mean absolute error based on the sklearn.metrics.mean_absolute_error function.

More details here : Mean absolute error

transparentai.models.evaluation.regression.**mean_gamma_deviance**(*y_true*, *y_pred*, ***args*)

Mean Gamma deviance based on the sklearn.metrics.mean_gamma_deviance function.

More details here : Mean Poisson, Gamma, and Tweedie deviances

transparentai.models.evaluation.regression.**mean_poisson_deviance**(*y_true*, *y_pred*, ***args*)

Mean Poisson deviances based on the sklearn.metrics.mean_poisson_deviance function.

More details here : Mean Poisson, Gamma, and Tweedie deviances

transparentai.models.evaluation.regression.**mean_squared_error**(*y_true*, *y_pred*, ***args*)

Mean squared error based on the sklearn.metrics.mean_squared_error function.

More details here : Mean squared error

transparentai.models.evaluation.regression.**mean_squared_log_error**(*y_true*, *y_pred*, ***args*)

Mean squared logarithmic error based on the sklearn.metrics.mean_squared_log_error function.

More details here : Mean squared logarithmic error

transparentai.models.evaluation.regression.**median_absolute_error**(*y_true*, *y_pred*, ***args*)

Median absolute error based on the sklearn.metrics.median_absolute_error function.

More details here : Median absolute error

transparentai.models.evaluation.regression.**r2**(*y_true*, *y_pred*, ***args*)

R^2 score, the coefficient of determination based on the sklearn.metrics.r2_score function.

More details here : R^2 score, the coefficient of determination

transparentai.models.evaluation.regression.**root_mean_squared_error**(*y_true*, *y_pred*, ***args*)

Root mean squared error based on the sklearn.metrics.mean_squared_error function.

squared argument is set to False.

More details here : Mean squared error

1.6.4 Model Explainer

class transparentai.models.explainers.**ModelExplainer**(*model*, *X*, *model_type=None*, *feature_names=None*, *multi_label=False*)

Class that allows to understand a local prediction or model behavior using shap package. For the moment this class will work only for model that TreeExplainer, LinearExplainer or KernelExplainer of shap package can handle.

Example

For binary classification (adult dataset):

```
>>> from transparentai.datasets import load_adult
>>> from sklearn.ensemble import RandomForestClassifier
```

```
>>> data = load_adult()
>>> X, Y = data.drop(columns='income'), data['income'].replace({'>50K':1, '<=50K
↪':0})
>>> X = X.select_dtypes('number')
>>> clf = RandomForestClassifier().fit(X,Y)
```

```
>>> explainer = ModelExplainer(clf, X, model_type='tree')
>>> explainer.explain_global_influence(X, nsamples=1000)
 99%|==================| 1988/2000 [01:01<00:00]
{'age': 0.08232147427281439,
 'fnlwgt': 0.051546309804410356,
 'education-num': 0.07579739409175655,
 'capital-gain': 0.07904473020037411,
 'capital-loss': 0.02746167321242212,
 'hours-per-week': 0.060904331971380544}
```

```
>>> explainer.explain_local_influence(X.iloc[0])
{'age = 25': -0.07041555656760465,
 'fnlwgt = 226802': -0.025452222766471095,
```

(continues on next page)

(continued from previous page)

```
 'education-num = 7': -0.07771055672375951,
 'capital-gain = 0': -0.08661166294186842,
 'capital-loss = 0': 0.005169999992358498,
 'hours-per-week = 40': -0.02528000040561892}
```

For multi label classification (iris dataset):

```
>>> from transparentai.datasets import load_iris
>>> from sklearn.ensemble import RandomForestClassifier
```

```
>>> data = load_iris()
>>> X, Y = data.drop(columns='iris plant'), data['iris plant']
>>> Y = Y.replace({'setosa':0, 'versicolor':1, 'virginica':2})
>>> clf = RandomForestClassifier().fit(X,Y)
```

```
>>> explainer = ModelExplainer(clf, X, model_type='tree', multi_label=True)
>>> explainer.explain_global_influence(X)
{0: {'sepal length (cm)': 0.01175688849131886,
  'sepal width (cm)': 0.005942666575467832,
  'petal length (cm)': 0.22338177293802924,
  'petal width (cm)': 0.16601288524931274},
 1: {'sepal length (cm)': 0.02572877729050815,
  'sepal width (cm)': 0.008901222137936085,
  'petal length (cm)': 0.2281212172475334,
  'petal width (cm)': 0.19257521807807496},
 2: {'sepal length (cm)': 0.02847011091114645,
  'sepal width (cm)': 0.011024999958494059,
  'petal length (cm)': 0.23041677331694704,
  'petal width (cm)': 0.20166499567956975}}
```

```
>>> explainer.explain_local_influence(X.iloc[0])
{0: {'sepal length (cm) = 5.1': 0.021333332546055316,
  'sepal width (cm) = 3.5': 0.011599999857135118,
  'petal length (cm) = 1.4': 0.42903332408517597,
  'petal width (cm) = 0.2': 0.31883332636673},
 1: {'sepal length (cm) = 5.1': 0.012099999799393118,
  'sepal width (cm) = 3.5': 0.0018000002391636372,
  'petal length (cm) = 1.4': -0.21319999573752285,
  'petal width (cm) = 0.2': -0.15029999669175595},
 2: {'sepal length (cm) = 5.1': -0.03343333344208076,
  'sepal width (cm) = 3.5': -0.013400000038091093,
  'petal length (cm) = 1.4': -0.21583332964917645,
  'petal width (cm) = 0.2': -0.16853333076229318}}
```

For regression (boston dataset):

```
>>> from transparentai.datasets import load_boston
>>> from sklearn.linear_model import LinearRegression
```

```
>>> data = load_boston()
>>> X, Y = data.drop(columns='MEDV'), data['MEDV']
>>> regr = LinearRegression().fit(X, Y)
```

```
>>> explainer = ModelExplainer(regr, X, model_type='linear')
>>> explainer.explain_global_influence(X)
```

(continues on next page)

(continued from previous page)

```
{'CRIM': 0.5167422492788898,
 'ZN': 0.7756203068845728,
 'INDUS': 0.12750516344183324,
 'CHAS': 0.3459732772883547,
 'NOX': 1.7001686711898643,
 'RM': 1.9555806154096416,
 'AGE': 0.017036261147963947,
 'DIS': 2.537086257135257,
 'RAD': 2.3074416123109764,
 'TAX': 1.7711676384532529,
 'PTRATIO': 1.7028349208723197,
 'B': 0.5086851450326836,
 'LSTAT': 2.9991432546436037}
```

```
>>> explainer.explain_local_influence(X.iloc[0])
{'CRIM = 0.0063': 0.3896189542190243,
 'ZN = 18.0': 0.308063041889274,
 'INDUS = 2.31': -0.18146644441613213,
 'CHAS = 0.0': -0.18584127208907195,
 'NOX = 0.538': 0.29661462781287745,
 'RM = 6.575': 1.1062538448823005,
 'AGE = 65.2': -0.002336189759535761,
 'DIS = 4.09': -0.4352292308278341,
 'RAD = 1.0': -2.616541593062981,
 'TAX = 296.0': 1.3843997187946957,
 'PTRATIO = 15.3': 3.006425898946704,
 'B = 396.9': 0.37457147693105614,
 'LSTAT = 4.98': 4.026504219585754}
```

model_type
Type of model to inspect, it can only be 'tree', 'linear' or None

model
model to inspect

multi_label
Whether there is more than 2 classes in the label column (only for classification)

Type bool

explainer
explainer object that has expected values and can compute shap values

Type shap.TreeExplainer, shap.LinearExplainer or shap.KernelExplainer

feature_names
list of feature names (length == length of X columns)

Type np.array

global_explain
dictionnary with feature names as keys and global feature importance as values

Type dict

Parameters

- **model** – model to inspect
- **X** (*array like*) – data (possibly training) to start the explainer

- **model_type** (*str (default None)*) – Type of model to inspect, it can only be 'tree', 'linear' or None
- **feature_names** (*np.array or list*) – list of feature names (length == length of X columns)
- **multi_label** (*bool*) – Whether there is more than 2 classes in the label column (only for classification)

Raises

- TypeError: – X must be an array like. Valids dtypes are pandas.DataFrame, pandas.Series, numpy.ndarray and list
- ValueError: – model_type must be 'tree', 'linear' or None
- AttributeError: – model has neither a predict() function or a predict_proba() function

compute_shap_values(*X*)

Computes the shap values using explainer attribute.

Parameters **X** (*array like*) – A matrix of samples (# samples x # features) on which to explain the model's output.

Returns For models with a single output this returns a matrix of SHAP values (# samples x # features). Each row sums to the difference between the model output for that sample and the expected value of the model output (which is stored as expected_value attribute of the explainer). For models with vector outputs this returns a list of such matrices, one for each output.

Return type np.ndarray

explain_global_influence(*X*, *nsamples=None*)

Global explaination for a model based on a sample X If there are a lot of data this function could last a while.

Parameters

- **X** (*array like*) – Data to explain
- **nsamples** (*None or int or float (default None)*) – If not None reduce the data to a sample of nsamples else if <= 1. reduce to len(df) * nsamples

Returns dictionnary with feature names as keys and feature importance as values

Return type dict

Raises

- TypeError: – X must be an array like. Valids dtypes are pandas.DataFrame, pandas.Series, numpy.ndarray and list
- ValueError: – X must have the same number of feature than the X used in the class initialization

explain_local_influence(*X*, *feature_classes=None*)

Explain a local prediction : only one row required.

Parameters

- **X** (*array like*) – Local row to explain
- **feature_classes** (*dict*) – This dictionnary provides new values for categorical feature so that the feature can be more interpretable. dictionnary with features names as keys

and for value a dictionnary with key, value pair representing current value and value to display.

Returns dictionnary with feature names as keys and feature importance as values

Return type dict

Raises

- TypeError: – X must be an array like. Valids dtypes are pandas.DataFrame, pandas.Series, numpy.ndarray and list
- ValueError: – X must be one row
- ValueError: – X must have the same number of feature than the X used in the class initialization

format_feature_importance(*feat_importance*, *top=None*)

Format feature importance with a top value so that it returns only the features that have the biggest influence

Parameters

- **feat_importance** (*pd.Series or dict*) – current feature importance
- **top** (*int*) – number of value to get

Returns Feature importance formated

Return type pd.Series

init_explainer(*X*)

Initialize the explainer.

If model_type is None then use shap.KernelExplainer class.

Else if it's 'tree' then shap.TreeExplainer.

Else use shap.LinearExplainer.

Parameters **X** (*array like*) – data (possibly training) to start the explainer

Returns explainer initialized

Return type shap.KernelExplainer, shap.TreeExplainer, shap.LinearExplainer

plot_global_explain(*X=None*, *nsamples=None*, *top=None*, *color='#3498db'*, ***kwargs*)

Display a plot for model global explanation based on a sample X.

Parameters

- **X** (*pd.DataFrame or np.array*) – Data to explain
- **nsamples** (*None or int or float (default None)*) – If not None reduce the data to a sample of nsamples else if <= 1. reduce to len(df) * nsamples
- **top** (*int*) – top n feature to display (in case there are too much features)
- **str (default '#3498db')** (*color*) – Color of the bar plot

Raises AttributeError: – If X parameter is None then you have to add X in explain_global function first or directly in this function if you prefer to plot directly.

plot_local_explain(*X*, *feature_classes=None*, *top=None*, *num_class=None*, ***kwargs*)

Display a plot for a local prediction based on X set.

Parameters

- **X** (*array like*) – Local row to explain

- **feature_classes** (*dict*) – This dictionnary provides new values for categorical feature so that the feature can be more interpretable. dictionnary with features names as keys and for value a dictionnary with key, value pair representing current value and value to display.
- **num_class** (*int (default None)*) – Class number for which we want to see the explanation if it's a binary classification then the value is 1 if None and it's a multi_label classifier then plots for each class

plot_local_explain_interact(*X*, *feature_classes=None*, *visible_feat=None*, *top=None*, *num_class=None*, ***kwargs*)

Display a plot for a local prediction based on X set.

Parameters

- **X** (*array like*) – Local row to explain
- **feature_classes** (*dict*) – This dictionnary provides new values for categorical feature so that the feature can be more interpretable. dictionnary with features names as keys and for value a dictionnary with key, value pair representing current value and value to display.
- **visible_feat** (*list (default None)*) – List of feature to interact with
- **num_class** (*int (default None)*) – Class number for which we want to see the explanation if it's a binary classification then the value is 1 if None and it's a multi_label classifier then plots for each class

1.7 transparentai.monitoring

1.7.1 Monitoring submodule

transparentai.monitoring.monitoring.**compute_metrics_groupby**(*y_true*, *y_pred*, *groupby*, *metrics*, *classification*)

Computes metrics groupby an array.

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like (1D or 2D)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)
- **groupby** (*array like*) – Array of values to groupby the computed metrics by
- **metrics** (*list*) – List of metrics to compute
- **classification** (*bool*) – Whether the ML task is a classification or not

Returns DataFrame with groubpy values as indexes and computed metrics as columns

Return type pd.DataFrame

transparentai.monitoring.monitoring.**monitor_model**(*y_true*, *y_pred*, *timestamp=None*, *interval='month'*, *metrics=None*, *classification=False*)

Monitor model over a timestamp array which represent the date or timestamp of the prediction.

If timestamp is None or interval then it just compute the metrics on all the predictions.

If interval is not None it can be one of the following : 'year', 'month', 'day' or 'hour'.

- 'year' : format '%Y'
- 'month' : format '%Y-%m'
- 'day' : format '%Y-%m-%d'
- 'hour' : format '%Y-%m-%d-%r'

If it's for a classification and you're using y_pred as probabilities don't forget to pass the classification=True argument !

You can use your choosing metrics. for that refer to the *evaluation metrics* documentation.

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like (1D or 2D)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)
- **timestamp** (*array like or None (default None)*) – Array of datetime when the prediction occured
- **interval** (*str or None (default 'month')*) – interval to format the timestamp with
- **metrics** (*list (default None)*) – List of metrics to compute
- **classification** (*bool (default True)*) – Whether the ML task is a classification or not

Returns DataFrame with datetime interval as indexes and computed metrics as columns

Return type pd.DataFrame

Raises

- ValueError: – interval must be 'year', 'month', 'day' or 'hour'
- TypeError: – y_true must be an array like
- TypeError: – timestamp must be an array like

1.8 transparentai.sustainable

This submodule contains functions in the *transparentai.sustainable* submodule.

1.8.1 Sustainable submodule

transparentai.sustainable.energy_usage.**evaluate_kWh**(*func*, **args*, *verbose=False*)

Using energyusage.evaluate function returns the result of the function and the effective emissions of the function (in kWh)

With verbose = True you can see the report with details.

If you want a pdf please use the following:

```
>>> energyusage.evaluate(func, *args, pdf=True)
```

From energyusage package.

Parameters

- **func** – User's function
- **verbose** (*bool (default False)*) – Whether it shows details or not

Returns

- *float* – effective emissions of the function in kWh
- *any* – function's return

transparentai.sustainable.sustainable.**emissions**(*process_kwh*, *breakdown*, *location*)

Calculates the CO2 emitted by the program based on the location

Parameters

- **process_kwh** (*int*) – kWhs used by the process
- **breakdown** (*list*) – energy mix corresponding to user's location
- **location** (*str*) – location of user

Returns emission in kilograms of CO2 emitted

Return type float

Raises ValueError: – Process wattage must be greater than 0.

transparentai.sustainable.sustainable.**energy_mix**(*location*)

Gets the energy mix information for a specific location

Parameters

- **location** (*str*) – user's location
- **location_of_default** (*str*) – Specifies which average to use if location cannot be determined

Returns percentages of each energy type

Return type list

Raises ValueError: – location must be a valid countries

transparentai.sustainable.sustainable.**estimate_co2**(*hours*, *location*, *watts=250*, *powerLoss=0.8*)

Returns co2 consumption in kg CO2

To find out the wattage of the machine used for training, I recommend you use this website: Newegg's Power Supply Calculator .

Based on this website: Power Management Statistics we can estimate an average wattage to be 250 Watts, but be carefull, it's only an estimation. So if you're using a computer with GPU or others components I recommend you use the first website that allows you to compute your wattage.

Parameters

- **hours** (*int*) – time of training in hours
- **location** (*str*) – location of user
- **watts** (*int (default 250)*) – Wattage of the computer or server that was used for training
- **powerLoss** (*float (default 0.8)*) – PSU efficiency rating

Returns emission in kilograms of CO2 emitted

Return type float

transparentai.sustainable.sustainable.**get_energy_data**(*year=2016*)
Loads enery data from a specify year (only 2016 is currently available)

Parameters **year** (*int (default 2016)*) – Year of the energy mix data

Returns Energy mix per country of the selected year

Return type dict

1.9 transparentai.security

This submodule contains functions in the *transparentai.security* submodule.

1.9.1 Security submodule

transparentai.security.safety.**check_packages_security**(*full_report=True*)
Using safety package, check out the known vulnerabilities of the installed packages.

For more details you can look at the package page : https://github.com/pyupio/safety

Parameters **full_report** (*True*) – Whether you want the full report or short report.

1.10 transparentai.utils

This submodule contains utility functions for *transparentai* module.

1.10.1 Reports functions

transparentai.utils.reports.**generate_head_page**(*document_title*)
Generate a figure with a given title.

Parameters **document_title** (*str*) – Name of the document

Returns Document head figure

Return type matplotlib.figure.Figure

transparentai.utils.reports.**generate_validation_report**(*model*, *X*, *y_true*, *X_valid=None*, *y_true_valid=None*, *metrics=None*, *model_type='classification'*, *out='validation_report.pdf'*)
Generate a pdf report on the model performance with the following graphics:

- First page with the report title
- An histogram of the y_true distribution
- Model performance plot
- Model feature importance plot

This function is usefull to keep a proof of the validation.

Parameters

- **model** – Model to analyse
- **X** (*array like*) – Features
- **y_true** (*array like*) – True labels
- **X_valid** (*array like*) – Features for validation set
- **y_true_valid** (*array like (default None)*) – True labels for validation set
- **metrics** (*list (default None)*) – List of metrics to plots
- **model_type** (*str (default 'classification')*) – 'classification' or 'regression'
- **out** (*str (default 'validation_report.pdf')*) – path where to save the report

Raises ValueError: – 'model_type must be 'classification' or 'regression'

1.10.2 Utility functions

transparentai.utils.utils.**encode_categorical_vars**(*df*)

Encodes categorical variables from a dataframe to be numerical (discrete) It uses LabelEncoder classes from scikit-learn

Parameters **df** (*pd.DataFrame*) – Dataframe to update

Returns

- *pd.DataFrame* – Encoded dataframe
- *dict* – Encoders with feature name on keys and encoder as value

transparentai.utils.utils.**find_dtype**(*arr*, *len_sample=1000*)

Find the general dtype of an array. Three possible dtypes :

- Number
- Datetime
- Object

Parameters

- **arr** (*array-like*) – Array to inspect
- **len_sample** (*int (default, 1000)*) – Number max of items to analyse if len_sample > len(arr) then use len(arr)

Returns dtype string ('number', 'datetime' or 'object')

Return type str

Raises TypeError: – arr is not an array like

transparentai.utils.utils.**format_describe_str**(*desc*, *max_len=20*)

Returns a formated list for the matplotlib table cellText argument.

Each element of the list is like this : ['key ','value ']

Number of space at the end of the value depends on len_max argument.

Parameters

- **desc** (*dict*) – Dictionnary returned by the variable.describe function

- **len_max** (*int (default 20)*) – Maximum length for the values

Returns Formated list for the matplotlib table cellText argument

Return type list(list)

transparentai.utils.utils.**init_corr_matrix**(*columns*, *index*, *fill_diag=1.0*)

Returns a matrix n by m fill of 0 (except on the diagonal if squared matrix) Recommended for correlation matrix

Parameters

- **columns** – list of column names
- **index** – list of index names
- **fill_diag** (*float (default 1.)*) – if squared matrix, then set diagonal with this value

Returns Initialized matrix

Return type pd.DataFrame

transparentai.utils.utils.**is_array_like**(*obj*, *n_dims=1*)

Returns whether an object is an array like. Valid dtypes are list, np.ndarray, pd.Series, pd.DataFrame.

Parameters

- **obj** – Object to inspect
- **n_dims** (*int (default 1)*) – number of dimension accepted

Returns Whether the object is an array like or not

Return type bool

transparentai.utils.utils.**preprocess_metrics**(*input_metrics*, *metrics_dict*)

Preprocess the inputed metrics so that it maps with the appropriate function in metrics_dict global variable.

input_metrics can have str or function. If it's a string then it has to be a key from metrics_dict global variable dict

Returns a dictionnary with metric's name as key and metric function as value

Parameters

- **input_metrics** (*list*) – List of metrics to compute
- **metrics_dict** (*dict*) – Dictionnary to compare input_metrics with

Returns Dictionnary with metric's name as key and metric function as value

Return type dict

Raises TypeError: – input_metrics must be a list

1.11 transparentai.plots

All ploting functions in different submodules.

1.11.1 Common plots functions

transparentai.plots.plots.**plot_or_figure**(*fig*, *plot=True*)

Parameters

- **fig** (*matplotlib.figure.Figure*) – figure to plot or to returns
- **plot** (*bool (default True)*) – Whether you want to plot a figure or return it

Returns Figure

Return type matplotlib.figure.Figure

transparentai.plots.plots.**plot_table_score**(*perf*)

Insert a table of scores on a matplotlib graphic

Parameters **perf** (*dict*) – Dictionnary with computed score

1.11.2 Datasets variable plots functions

transparentai.datasets.variable.variable_plots.**plot_datetime_var**(*ax*, *arr*, *color='#3498db'*, *label=None*, *alpha=1.0*)

Plots a line plot into an matplotlib axe.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **arr** (*array like*) – Array of datetime values
- **color** (*str (default DEFAULT_COLOR)*) – color of the plot
- **label** (*str (default None)*) – label of the plot
- **alpha** (*float (default 1.)*) – opacity

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not a datetime array

transparentai.datasets.variable.variable_plots.**plot_number_var**(*ax*, *arr*, *color='#3498db'*, *label=None*, *alpha=1.0*)

Plots an histogram into an matplotlib axe.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **arr** (*array like*) – Array of number values
- **color** (*str (default DEFAULT_COLOR)*) – color of the plot
- **label** (*str (default None)*) – label of the plot
- **alpha** (*float (default 1.)*) – opacity

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not a number array

transparentai.datasets.variable.variable_plots.**plot_object_var**(*ax*, *arr*, *top=10*, *color='#3498db'*, *label=None*, *alpha=1.0*)

Plots a bar plot into an matplotlib axe.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **arr** (*array like*) – Array of object values
- **color** (*str (default DEFAULT_COLOR)*) – color of the plot
- **label** (*str (default None)*) – label of the plot
- **alpha** (*float (default 1.)*) – opacity

Raises

- TypeError: – arr is not an array like
- TypeError: – arr is not a object array

transparentai.datasets.variable.variable_plots.**plot_table_describe**(*ax*, *cell_text*)

Insert a table in a matplotlib graphic using an axis.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **cell_text** (*list(list)*) – The texts to place into the table cells.

transparentai.datasets.variable.variable_plots.**plot_variable**(*arr*, *legend=None*, *colors=None*, *xlog=False*, *ylog=False*, ***kwargs*)

Plots a graph with two parts given an array. First part is the plot custom plot depending on the array dtype. Second part is the describe statistics table.

First plot is:

- Histogram if dtype is number (using plot_number_var)
- Line plot if dtype is datetime (using plot_datetime_var)
- Bar plot if dtype is object (using plot_object_var)

If legend array is set then automaticly plots differents values.

Parameters

- **arr** (*array like*) – Array of values to plots
- **legend** (*array like (default None)*) – Array of values of legend (same length than arr)
- **colors** (*list (default None)*) – Array of colors, used if legend is set
- **xlog** (*bool (default False)*) – Scale xaxis in log scale
- **ylog** (*bool (default False)*) – Scale yaxis in log scale

Raises

- TypeError: – arr is not an array like

- TypeError: – legend is not an array like
- ValueError: – arr and legend have not the same length

1.11.3 Classification plots functions

transparentai.models.classification.classification_plots.**compute_prob_performance**(*y_true*, *y_prob*, *metrics*)

Computes performance that require probabilities

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **metrics** (*list*) – List of metrics to compute

Returns Dictionnary of metrics computed that requires probabilities. If no metrics need those then it returns None

Return type dict

Raises TypeError: – metrics must be a list

transparentai.models.classification.classification_plots.**plot_confusion_matrix**(*confusion_matrix*

Show confusion matrix.

Parameters **confusion_matrix** (*array*) – confusion_matrix metrics result

transparentai.models.classification.classification_plots.**plot_performance**(*y_true*, *y_pred*, *y_true_valid=None*, *y_pred_valid=None*, *metrics=None*, ***kwargs*)

Plots the performance of a classifier. You can use the metrics of your choice with the metrics argument

Can compare train and validation set.

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like (1D or 2D)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)
- **y_true_valid** (*array like (default None)*) – True labels
- **y_pred_valid** (*array like (1D or 2D) (default None)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)
- **metrics** (*list (default None)*) – List of metrics to plots

Raises TypeError: – if metrics is set it must be a list

transparentai.models.classification.classification_plots.**plot_roc_curve**(*roc_curve*, *roc_auc*)

Show a roc curve plot with roc_auc score on the legend.

Parameters

- **roc_curve** (*array*) – roc_curve metrics result for each class
- **roc_auc** (*array*) – roc_auc metrics result for each class

transparentai.models.classification.classification_plots.**plot_score_function**(*perf*, *perf_prob*, *metric*)

Plots score with a specific function.

E.g. confusion_matrix or roc_auc

Parameters

- **perf** (*dict*) – Dictionnary with computed score
- **perf_prob** (*dict*) – Dictionnary with computed score (using probabilities)
- **metric** (*str*) – name of the metric

Raises ValueError: – metric does not have a plot function

transparentai.models.classification.classification_plots.**plot_table_score_clf**(*perf*)

Insert a table of scores on a matplotlib graphic for a classifier

Parameters **perf** (*dict*) – Dictionnary with computed score

transparentai.models.classification.classification_plots.**preprocess_scores**(*y_pred*)

Preprocess y_pred for plot_performance function.

if y_pred is probabilities then y_pred become predicted class, y_prob is the probabilities else, y_prob is None

Parameters **y_pred** (*array like (1D or 2D)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)

Returns

- *np.ndarray* – array with predicted labels
- *np.ndarray* – array with probabilities if available else None
- *int* – number of classes

1.11.4 Regression plots functions

transparentai.models.regression.regression_plots.**plot_error_distribution**(*errors*)

Plots the error distribution with standard deviation, mean and median.

The error is calculated by the following formula :

$$error = y - \hat{y}$$

Parameters **errors** (*array like*) – Errors of a regressor

transparentai.models.regression.regression_plots.**plot_performance**(*y_true*, *y_pred*, *y_true_valid=None*, *y_pred_valid=None*, *metrics=None*, ***kwargs*)

Plots the performance of a regressor. You can use the metrics of your choice with the metrics argument

Can compare train and validation set.

Parameters

- **y_true** (*array like*) – True target values
- **y_pred** (*array like*) – Predicted values
- **y_true_valid** (*array like (default None)*) – True target values for validation set
- **y_pred_valid** (*array like (1D or 2D) (default None)*) – Predicted values for validation set
- **metrics** (*list*) – List of metrics to plots

Raises TypeError: – if metrics is set it must be a list

1.11.5 Explainer plots functions

transparentai.models.explainers.explainer_plots.**plot_global_feature_influence**(*feat_importance*, *color='#3498db'*, ***kwargs*)

Display global feature influence sorted.

Parameters **feat_importance** (*pd.Series*) – Feature importance with feature as indexes and shap value as values

transparentai.models.explainers.explainer_plots.**plot_local_feature_influence**(*feat_importance*, *base_value*, *pred*, *pred_class=None*, ***kwargs*)

Display local feature influence sorted for a specific prediction.

Parameters

- **feat_importance** (*pd.Series*) – Feature importance with feature as indexes and shap value as values
- **base_value** (*number*) – prediction value if we don't put any feature into the model
- **pred** (*number*) – predicted value

1.11.6 Fairness plots functions

transparentai.fairness.fairness_plots.**format_priv_text**(*values*, *max_char*)

Formats privileged (or unprivileged) values text so that it can be shown.

Parameters

- **values** (*list*) – List of privileged or unprivileged values
- **max_char** (*int*) – Maximum characters allow in the returned string

Returns Formated string for given values

Return type str

Raises TypeError – values must be a list

transparentai.fairness.fairness_plots.**get_protected_attr_values**(*attr*, *df*, *privileged_group*, *privileged=True*)

Retrieves all values given the privileged_group argument.

If privileged is True and privileged_group[attr] is a list then it returns the list, if it's a function then values of df[attr] for which the function returns True.

If privileged is False and privileged_group[attr] is a list then it returns values of df[attr] not in the list else if it's a function returns values of df[attr] for which the function returns False.

Parameters

- **attr** (*str*) – Protected attribute which is a key of the privileged_group dictionnary
- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group
- **privileged** (*bool (default True)*) – Boolean prescribing whether to condition this metric on the *privileged_groups*, if *True*, or the *unprivileged_groups*, if *False*.

Returns List of privileged values of the protected attribyte attr if privileged is True else unprivileged values

Return type list

Raises ValueError: – attr must be in privileged_group

transparentai.fairness.fairness_plots.**plot_attr_title**(*ax*, *attr*, *df*, *privileged_group*)

Plots the protected attribute titles with :

- The attribute name (e.g. Gender)
- Priviliged and unprivileged values
- Number of privileged and unprivileged values

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **attr** (*str*) – Protected attribute which is a key of the privileged_group dictionnary
- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group

Raises

- ValueError: – attr must be in df columns
- ValueError: – attr must be in privileged_group keys

transparentai.fairness.fairness_plots.**plot_bias**(*y_true*, *y_pred*, *df*, *privileged_group*, *pos_label=1*, *regr_split=None*, *with_text=True*, ***kwargs*)

Plots the fairness metrics for protected attributes refered in the privileged_group argument.

It uses the 4 fairness function :

- statistical_parity_difference
- disparate_impact
- equal_opportunity_difference
- average_odds_difference

You can also use it for a regression problem. You can set a value in the regr_split argument so it converts it to a binary classification problem. To use the mean use 'mean'. If the favorable label is more than the split value set pos_label argument to 1 else to 0.

Example

Using this function for a binary classifier:

```
>>> from transparentai.datasets import load_adult
>>> from sklearn.ensemble import RandomForestClassifier
```

```
>>> data = load_adult()
>>> X, Y = data.drop(columns='income'), data['income'].replace({'>50K':1, '<=50K
↪':0})
>>> X = X.select_dtypes('number')
>>> clf = RandomForestClassifier().fit(X,Y)
>>> y_pred = clf.predict(X)
```

```
>>> privileged_group = { 'gender':['Male'] }
```

```
>>> y_pred = clf.predict(X)plot_bias(Y, y_pred, data, privileged_group, with_
↪text=True)
```

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like*) – Predicted labels
- **df** (*pd.DataFrame*) – Dataframe to extract privilieged group from.
- **privileged_group** (*dict*) – Dictionnary with protected attribute as key (e.g. age or gender) and a list of favorable value (like ['Male']) or a function returning a boolean corresponding to a privileged group
- **pos_label** (*number*) – The label of the positive class.
- **regr_split** (*'mean' or number (default None)*) – If it's a regression problem then you can convert result to a binary classification using 'mean' or a choosen number. both y_true and y_pred become 0 and 1 : 0 if it's equal or less than the split value (the average if 'mean') and 1 if more. If the favorable label is more than the split value set pos_label=1 else pos_label=0
- **with_text** (*bool (default True)*) – Whether it displays the explanation text for fairness metrics.

transparentai.fairness.fairness_plots.**plot_bias_one_attr**(*ax*, *metric*, *score*)

Plots bias metric score bar with the indication if it's considered not fair or fair.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **metric** (*str*) – The name of the metric
- **score** (*float:*) – Score value of the metric

transparentai.fairness.fairness_plots.**plot_fairness_text**(*ax*, *score*, *metric*)

Plots bias metric explanation text.

The text is retrieved by the fairness_metrics_text() function.

Parameters

- **ax** (*plt.axes.Axes*) – axe where to add the plot
- **metric** (*str*) – The name of the metric
- **score** (*float:*) – Score value of the metric

1.11.7 Monitoring plots functions

transparentai.monitoring.monitoring_plots.**plot_monitoring**(*y_true*, *y_pred*, *timestamp=None*, *interval='month'*, *metrics=None*, *classification=False*, ***kwargs*)

Plots model performance over a timestamp array which represent the date or timestamp of the prediction.

If timestamp is None or interval then it just compute the metrics on all the predictions.

If interval is not None it can be one of the following : 'year', 'month', 'day' or 'hour'.

- 'year' : format '%Y'
- 'month' : format '%Y-%m'
- 'day' : format '%Y-%m-%d'
- 'hour' : format '%Y-%m-%d-%r'

If it's for a classification and you're using y_pred as probabilities don't forget to pass the classification=True argument !

You can use your choosing metrics. for that refer to the *evaluation metrics* documentation.

Parameters

- **y_true** (*array like*) – True labels
- **y_pred** (*array like (1D or 2D)*) – if 1D array Predicted labels, if 2D array probabilities (returns of a predict_proba function)
- **timestamp** (*array like or None (default None)*) – Array of datetime when the prediction occured
- **interval** (*str or None (default 'month')*) – interval to format the timestamp with
- **metrics** (*list (default None)*) – List of metrics to compute
- **classification** (*bool (default True)*) – Whether the ML task is a classification or not

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

P

R

S

T