explabox

National Police Lab AI (NPAI)

Mar 18, 2024

USING EXPLABOX

1 Quick tour 3

2 Using explabox 9

3 Development 11

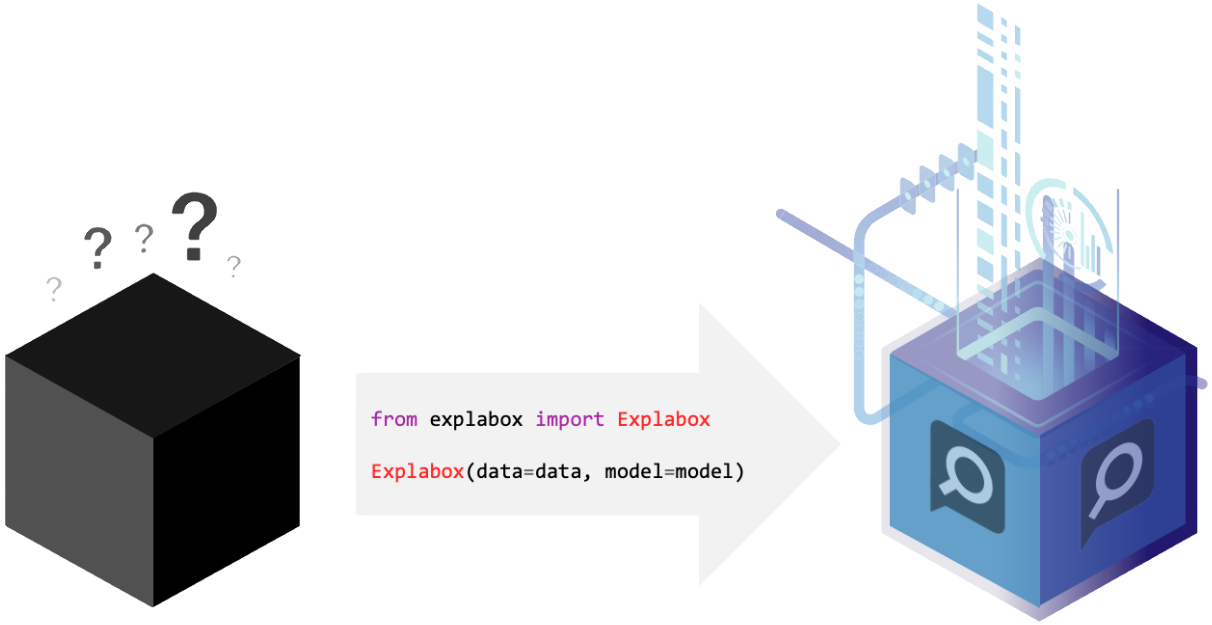
4 Citation 13

Python Module Index 53

Index 55

explabox

The `explabox` aims to support data scientists and machine learning (ML) engineers in explaining, testing and documenting AI/ML models, developed in-house or acquired externally. The `explabox` turns your **ingestibles** (AI/ML model and/or dataset) into **digestibles** (statistics, explanations or sensitivity insights)!



The `explabox` can be used to:

- **Explore**: describe aspects of the model and data.
- **Examine**: calculate quantitative metrics on how the model performs.

- **Expose**: see model sensitivity to random inputs (*safety*), test model generalizability (e.g. sensitivity to typos; *robustness*), and see the effect of adjustments of attributes in the inputs (e.g. swapping male pronouns for female pronouns; *fairness*), for the dataset as a whole (*global*) as well as for individual instances (*local*).
- **Explain**: use XAI methods for explaining the whole dataset (*global*), model behavior on the dataset (*global*), and specific predictions/decisions (*local*).

A number of experiments in the `explabox` can also be used to provide transparency and explanations to stakeholders, such as end-users or clients.

Note: The `explabox` currently only supports natural language text as a modality. In the future, we intend to extend to other modalities.

CHAPTER

ONE

QUICK TOUR

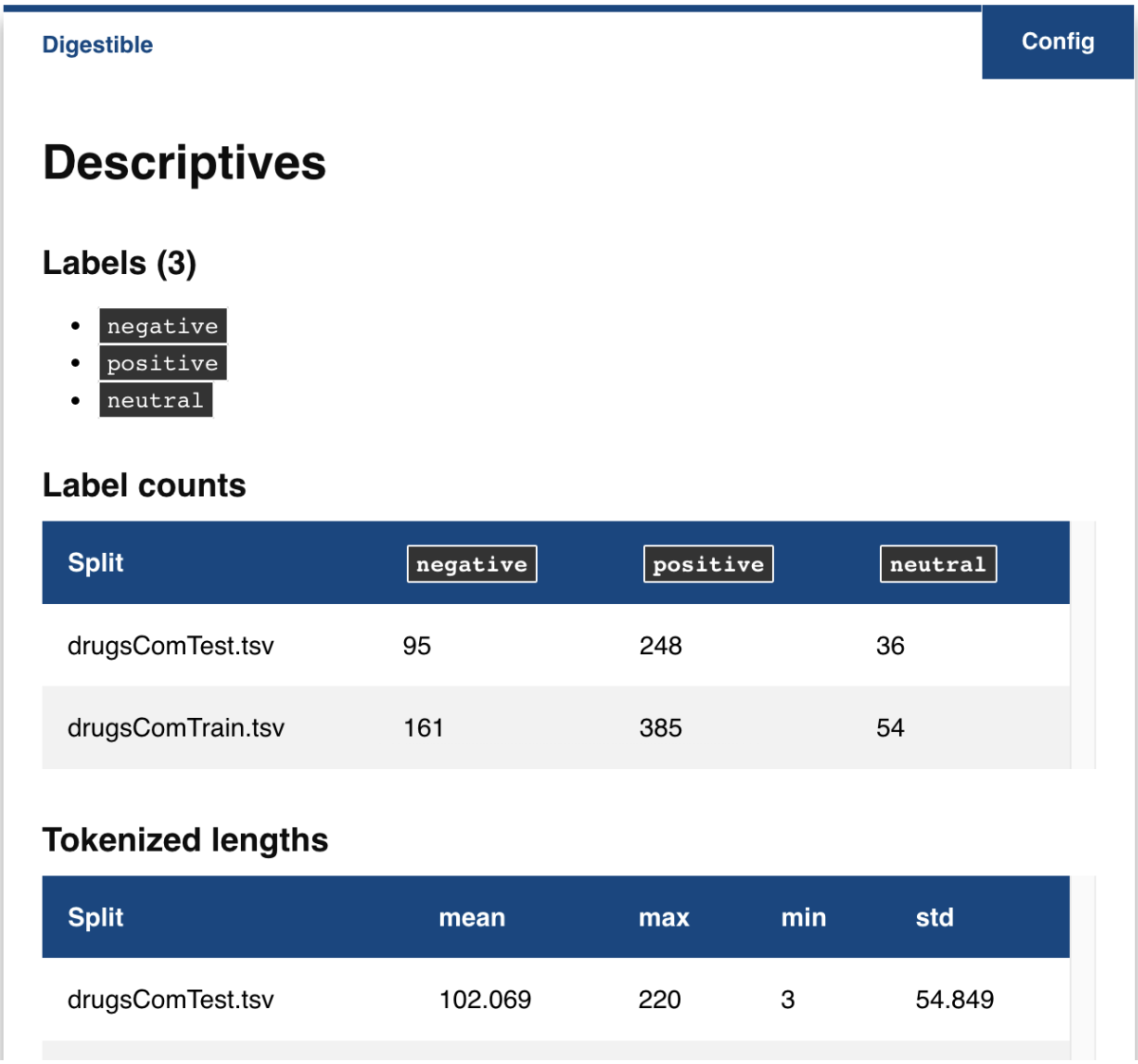
The `explabox` is distributed on PyPI. To use the package with Python, install it (`pip install explabox`), import your `data` and `model` and wrap them in the `Explabox`:

```
>>> from explabox import import_data, import_model
>>> data = import_data('./drugsCom.zip', data_cols='review', label_cols='rating')
>>> model = import_model('model.onnx', label_map={0: 'negative', 1: 'neutral', 2:
↪'positive'})

>>> from explabox import Explabox
>>> box = Explabox(data=data,
...                model=model,
...                splits={'train': 'drugsComTrain.tsv', 'test': 'drugsComTest.tsv'})
```

Then `.explore`, `.examine`, `.expose` and `.explain` your model:

```
>>> # Explore the descriptive statistics for each split
>>> box.explore()
```



Digestible

Config

Descriptives

Labels (3)

- `negative`
- `positive`
- `neutral`

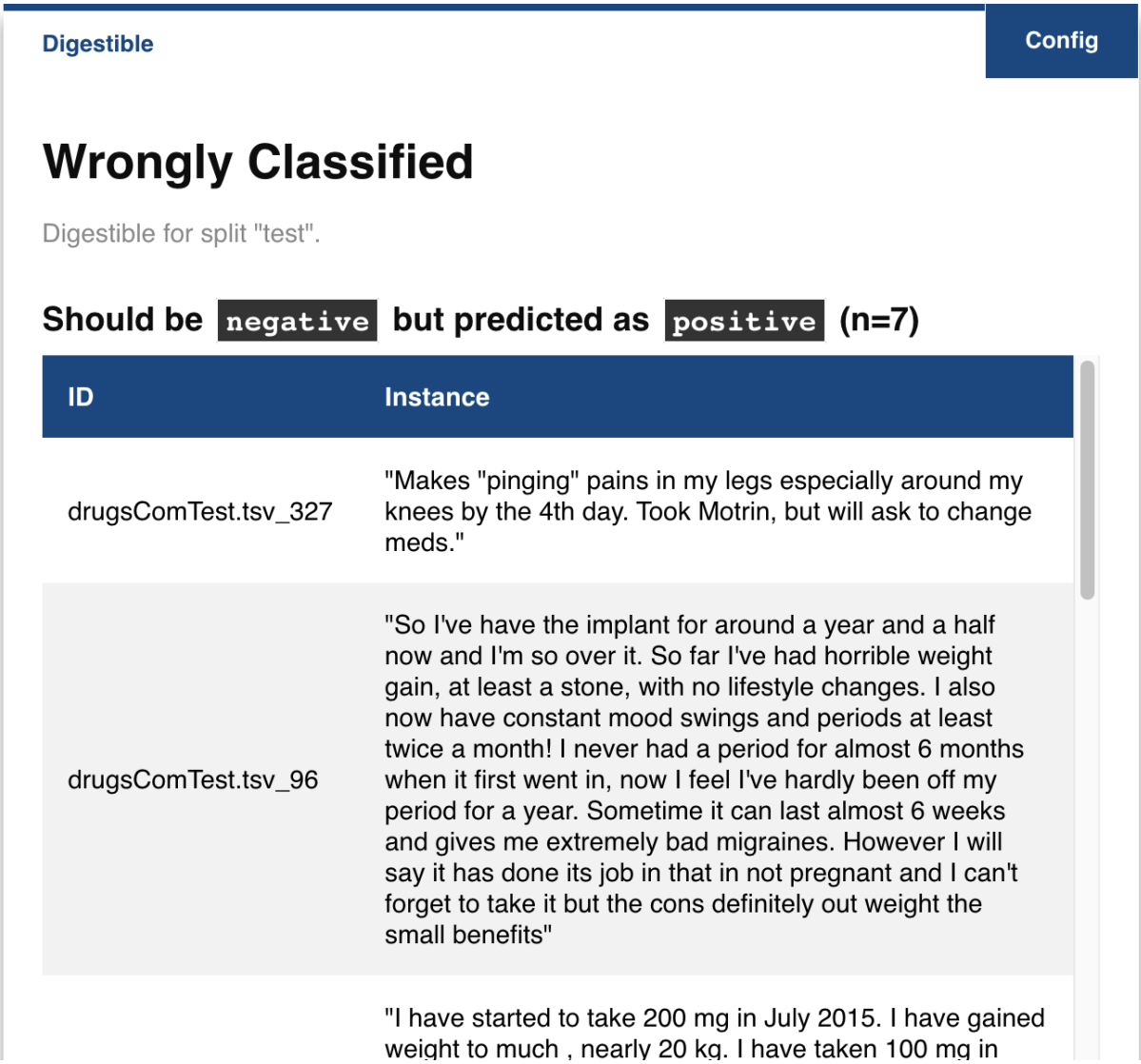
Label counts

Split	`negative`	`positive`	`neutral`
drugsComTest.tsv	95	248	36
drugsComTrain.tsv	161	385	54

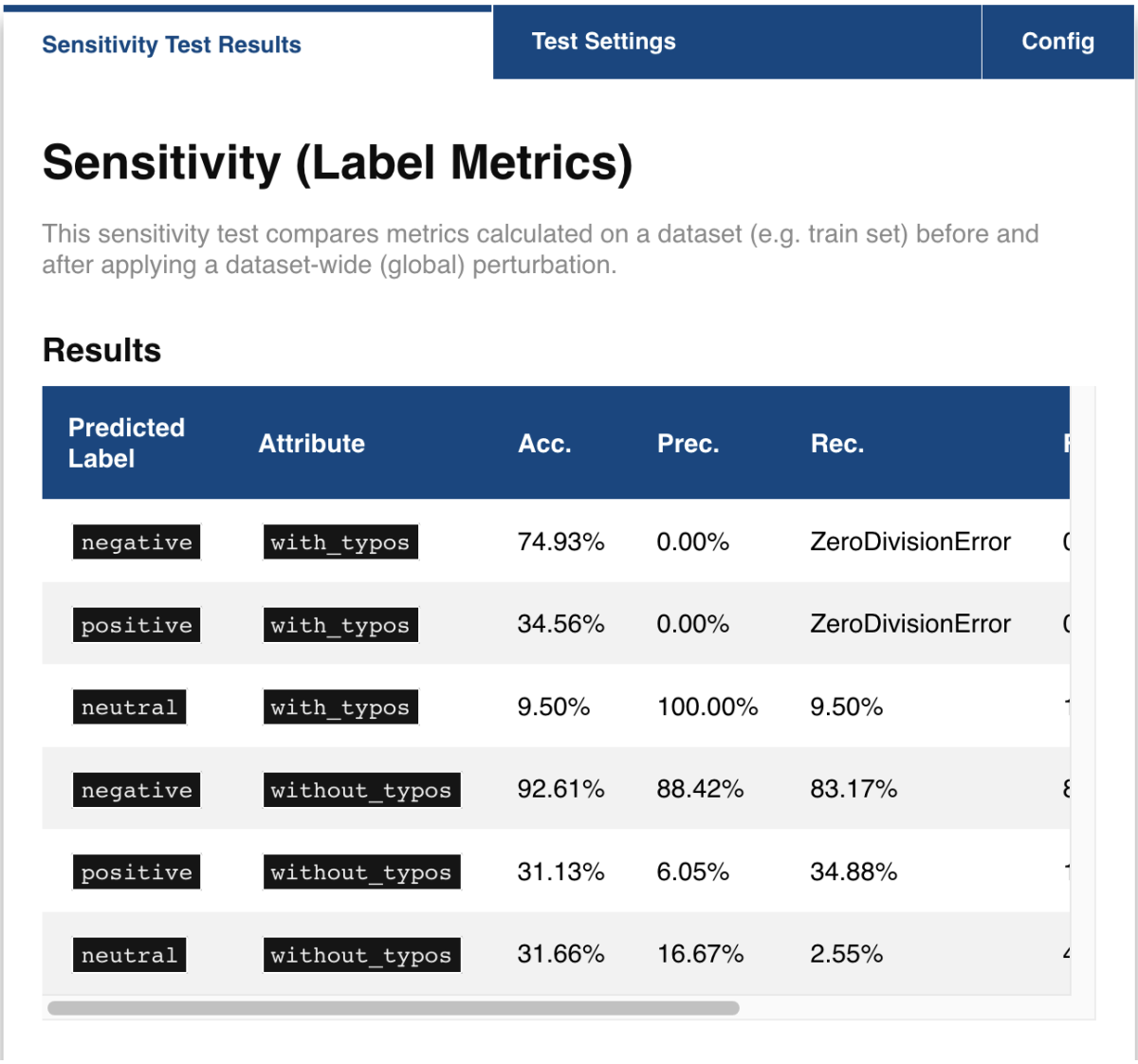
Tokenized lengths

Split	mean	max	min	std
drugsComTest.tsv	102.069	220	3	54.849

```
>>> # Show wrongly classified instances
>>> box.examine.wrongly_classified()
```




```
>>> # Compare the performance on the test split before and after adding typos to the text
>>> box.expose.compare_metrics(split='test', perturbation='add_typos')
```



Sensitivity Test Results | Test Settings | Config

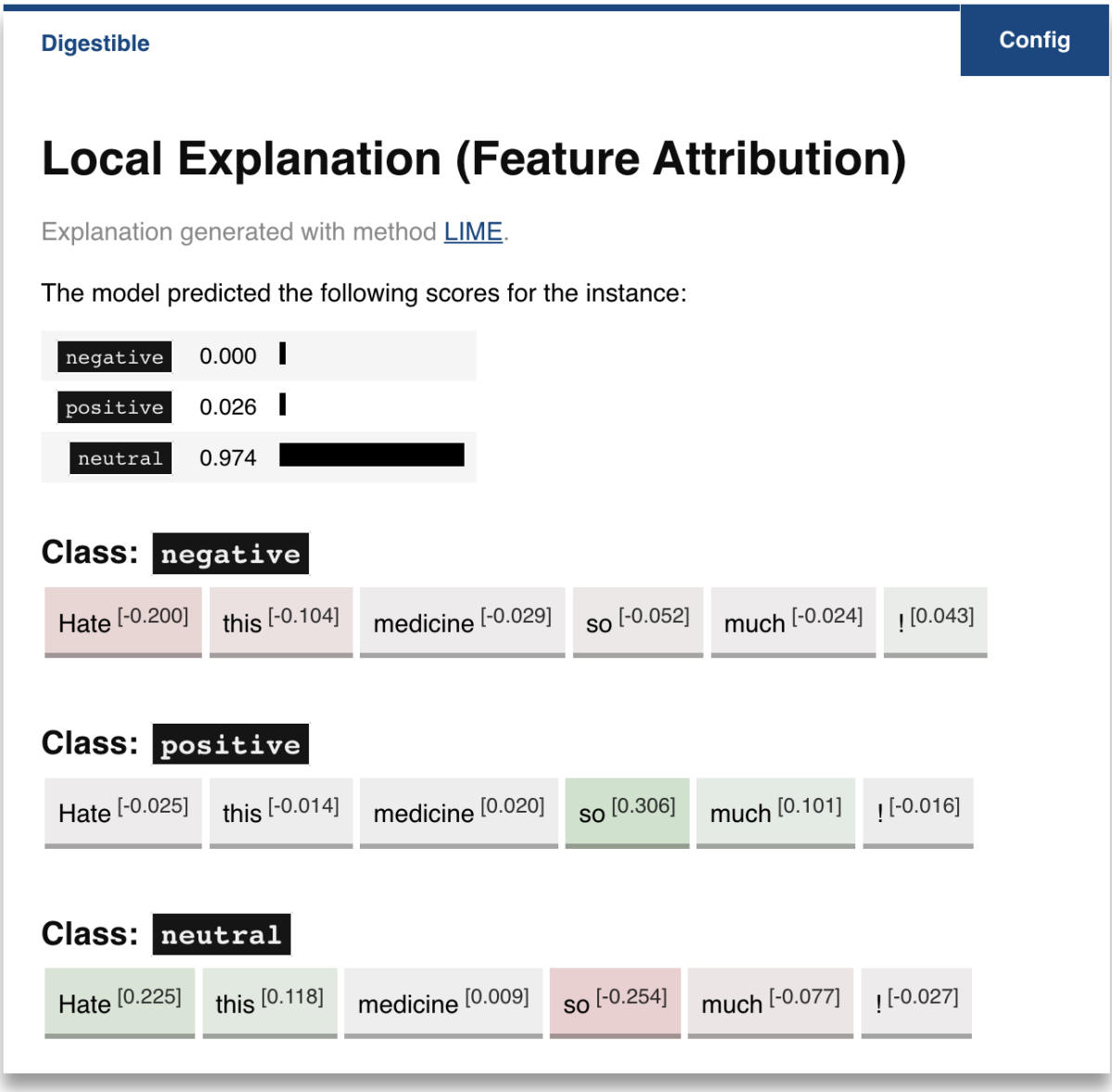
Sensitivity (Label Metrics)

This sensitivity test compares metrics calculated on a dataset (e.g. train set) before and after applying a dataset-wide (global) perturbation.

Results

Predicted Label	Attribute	Acc.	Prec.	Rec.	
negative	with_typos	74.93%	0.00%	ZeroDivisionError	
positive	with_typos	34.56%	0.00%	ZeroDivisionError	
neutral	with_typos	9.50%	100.00%	9.50%	
negative	without_typos	92.61%	88.42%	83.17%	
positive	without_typos	31.13%	6.05%	34.88%	
neutral	without_typos	31.66%	16.67%	2.55%	

```
>>> # Get a local explanation (uses LIME by default)
>>> box.explain.explain_prediction('Hate this medicine so much!')
```



Generated with explabox

CHAPTER

TWO

USING EXPLABOX

Installation
Installation guide, directly installing it via pip or through the git.

Example Usage
An extended usage example, showcasing how you can *explore*, *examine*, *expose* and *explain* your AI model.

Overview
Overview of the general idea behind the *explabox* and its package structure.

Explabox API reference
A reference to all classes and functions included in the *explabox*.

CHAPTER

THREE

DEVELOPMENT

Explabox @ GIT
: The git includes the open-source code and the most recent development version.

Changelog
: Changes for each version are recorded in the changelog.

Contributing
: A guide to making your own contributions to the open-source *explabox* package.

CHAPTER

FOUR

CITATION

...

4.1 Installation

Installation of the `explabox` requires `Python 3.8` or higher.

4.1.1 1. Python installation

Install Python on your operating system using the Python Setup and Usage guide.

4.1.2 2. Installing `explabox`

`explabox` can be installed:

1. *using* `pip`: `pip3 install` (released on PyPI)
2. *locally*: cloning the repository and using `python3 setup.py install`

Option 1: Using `pip`

1. Open up a `terminal` (Linux / macOS) or `cmd.exe/powershell.exe` (Windows)
2. Run the command:
    - `pip3 install explabox`, or
    - `pip install explabox`.

```
user@terminal:~$ pip3 install explabox
Collecting explabox
...
Installing collected packages: explabox
Successfully installed explabox
```

Option 2: Locally

1. Download the folder from `GitHub`:
    - Clone this repository, or
    - Download it as a `.zip` file and extract it.
2. Open up a `terminal` (Linux / macOS) or `cmd.exe/powershell.exe` (Windows) and navigate to the folder you downloaded `explabox` in.
3. In the main folder (containing the `setup.py` file) run:
    - `python3 setup.py install` / `python setup.py install` or,
    - `pip3 install .` / `pip install .`

```
user@terminal:~$ cd ~/explabox
user@terminal:~/explabox$ python3 setup.py install
running install
running bdist_egg
running egg_info
...
Finished processing dependencies for explabox
```

4.2 Example Usage

This page includes an extended example usage guide for using the `explabox` on a dataset and black-box model for multi-class classification.

4.2.1 Demo: Drug Review Classification

Welcome to the demo of the explabox on the UCI Drug Reviews dataset. To speed up the demo, we made a smaller subset of the train and test dataset. The demo also includes a pretrained black-box classifier, which aims to predict whether a `review` in the text got a `rating` of `negative` (1-5), `neutral` (5-6) or `positive` (6-10).

To start the demo, you require:

- Python 3.8 or above (see the Python installation guide)
- Jupyter Notebook installed (see the Jupyter installation guide)
- An active internet connection

Install the demo via:

- `pip3 install explabox`
- `pip3 install explabox-demo-drugreview`

Want to follow along?

We have prepared a Jupyter Notebook for you that runs all necessary lines for the demo.

> Having trouble? Want to know which functionalities the explabox includes? Check out the API reference at https://explabox.readthedocs.io/en/latest/api/explabox.html.

Let's get started!

The Jupyter Notebook demo will walk you through importing your data and model into the `explabox`, and go over some examples for the `explore`, `examine`, `expose` and `explain` parts. The demo is structured as follows:

1. ***Ingestibles***: importing your model and data.
2. ***Explore***: exploring the dataset descriptives.
3. ***Examine***: examine model behavior on the data.
4. ***Explain***: explain how the model behaves in general (global) and for specific predictions (local).
5. ***Expose***: find where your model is sensitive to, when it breaks down and when its behavior is unfair.
6. ***Challenges***: Challenges to further `explore`, `examine`, `expose` and `explain` your ingestibles and turn them into digestibles.

1. Ingestibles

The general idea of the `explabox` is to turn your `ingestibles` into `digestibles`. Your `ingestibles` (model and data) are typically opaque and difficult to understand. They require a lot of effort to trust and understand, to test, and communicate the results of these tests. We made that a lot simpler! We have several components that turn the `ingestibles` into `digestibles`: experiments that increase the transparency of your model, and are easy to understand and share.

To get started, the `explabox` requires a `model` (e.g. a Scikit-learn classifier or a deep neural network trained in PyTorch) and the `data` (e.g. CSV, HDF5, Huggingface datasets or a Pandas dataframe) you want to turn into `digestibles`. An example of a dataset for debugging purposes is the test set of the model. The `explabox` has two simple functions that import the data and model for you: `explabox.import_data(...)` and `explabox.import_model(...)`. For the demo, we have already provided you with the location of the dataset file (`dataset_file`) and imported the model for you (`model`):

```
from explabox_demo_drugreview import model, dataset_file
```

The UCI Drug Reviews dataset contains over 200,000 patient reviews for drugs. The dataset includes the following columns:

Column	Description	Data type
`drugName`	Name of drug discussed in review	Categorical (string)
`condition`	Name of condition discussed in review	Categorical (string)
`review`	Patient review	Free text (string)
`rating`	10 star patient rating	Rating `1-10` (integer)
`date`	Date of review entry	Date
`usefulCount`	Number of users who found the review useful	Continuous value >=`0` (integer)

To speed up the demo, we have included a small subset of examples from the original `train` and `test` set. We made a classifier, that predicts the sentiment for each review: `negative` (1-5), `neutral` (5-6) or `positive` (6-10). To make the performance comparisons easier, the `rating` column in the example dataset has also been transformed to these three categorical values.

Let us start by importing the data, where we are interesting in the textual reviews in the `'review'` column and the labels in the `'rating'` column. The `dataset_file` is the location of the dataset (`drugsCom.zip`), containing a train split (`drugsComTrain.tsv`) and test split (`drugsComTest.tsv`).

```
from explabox import import_data
data = import_data(dataset_file, data_cols='review', label_cols='rating')
```

The `model` included has already been passed through the `model = import_model(...)` function for you, and can therefore be used directly. This is on purpose, so the model is a true black-box for you!

Now let's explore/examine/expose/explain your model with the Explabox!

Make sure you explicitly include that `drugsComTrain.tsv` includes the train split and `drugsComTest.tsv` the test split of the data:

```
from explabox import Explabox

box = Explabox(data=data,
               model=model,
               splits={'train': 'drugsComTrain.tsv', 'test': 'drugsComTest.tsv'})
```

Now you are ready to `.explore`, `.examine`, `.expose` and `.explain` with the `explabox`!

NOTE

You can use `help(...)` at any time to better understand a model or function.

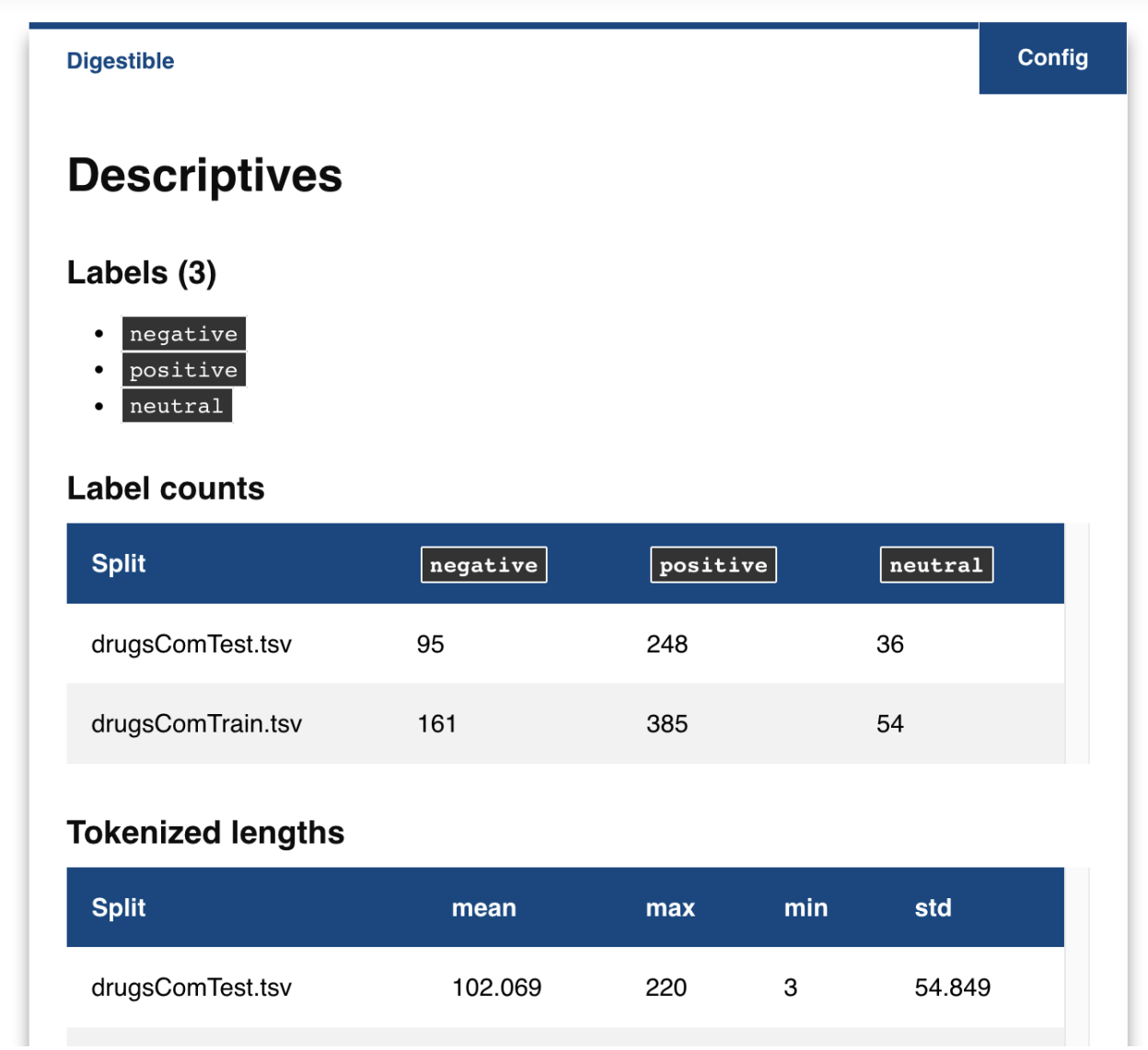
---

2. Explore

The *Explorer* explores your data by providing descriptive statistics. It is included in the Explabox under the `.explore` property.

Get descriptives for all splits by calling `box.explore()` or `box.explore.descriptives()`:

```
box.explore()
```



Digestible

Config

Descriptives

Labels (3)

- negative
- positive
- neutral

Label counts

Split	negative	positive	neutral
drugsComTest.tsv	95	248	36
drugsComTrain.tsv	161	385	54

Tokenized lengths

Split	mean	max	min	std
drugsComTest.tsv	102.069	220	3	54.849

Even though we trust you could have calculated each of these yourselves, it sure saves a lot of work. One simple call and that is all there is.

Want to look at a split of the data in more detail? Try `box.explore.instances()`:

```
dataset = box.explore.instances()
```

You can then further filter, drill-down or select you data. For instance by filtering on label 'positive'…

```
dataset.filter('positive')
```

… or taking the first 10 elements

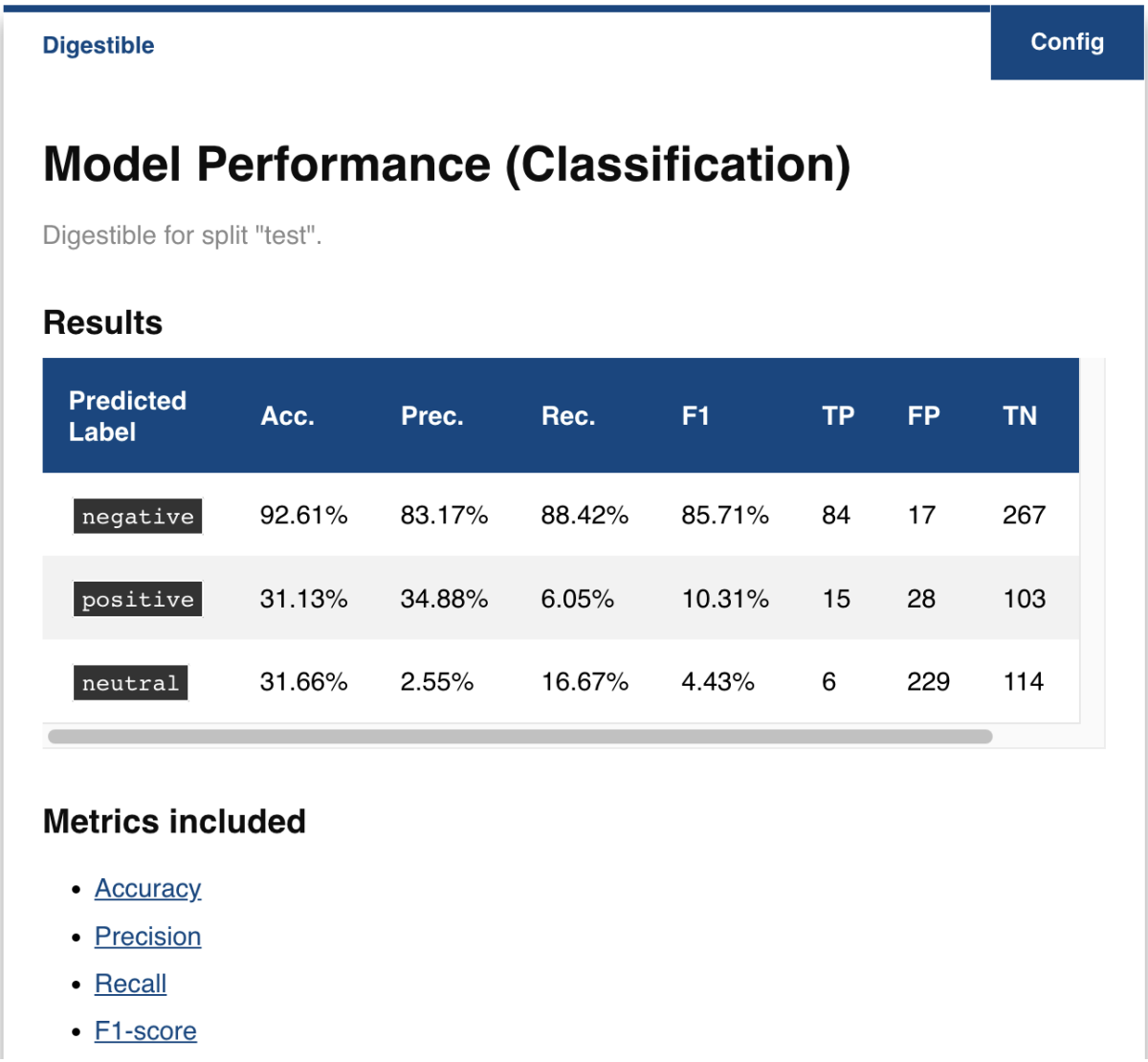
```
dataset.head(n=10)
```

Let's `examine` (see what I did there?) for some more impressive functionalities.

3. Examine

Now we've got a gist of what the data looks like, how does the model perform on the data? Simple, just call `box.examine()` or `box.examine.performance()`. To do so, the *Examiner* requires a 'model' and 'data'. It is included in the `explabox` under the `.examine` property.

```
box.examine(split='test')
```



Digestible

Config

Model Performance (Classification)

Digestible for split "test".

Results

Predicted Label	Acc.	Prec.	Rec.	F1	TP	FP	TN
negative	92.61%	83.17%	88.42%	85.71%	84	17	267
positive	31.13%	34.88%	6.05%	10.31%	15	28	103
neutral	31.66%	2.55%	16.67%	4.43%	6	229	114

Metrics included

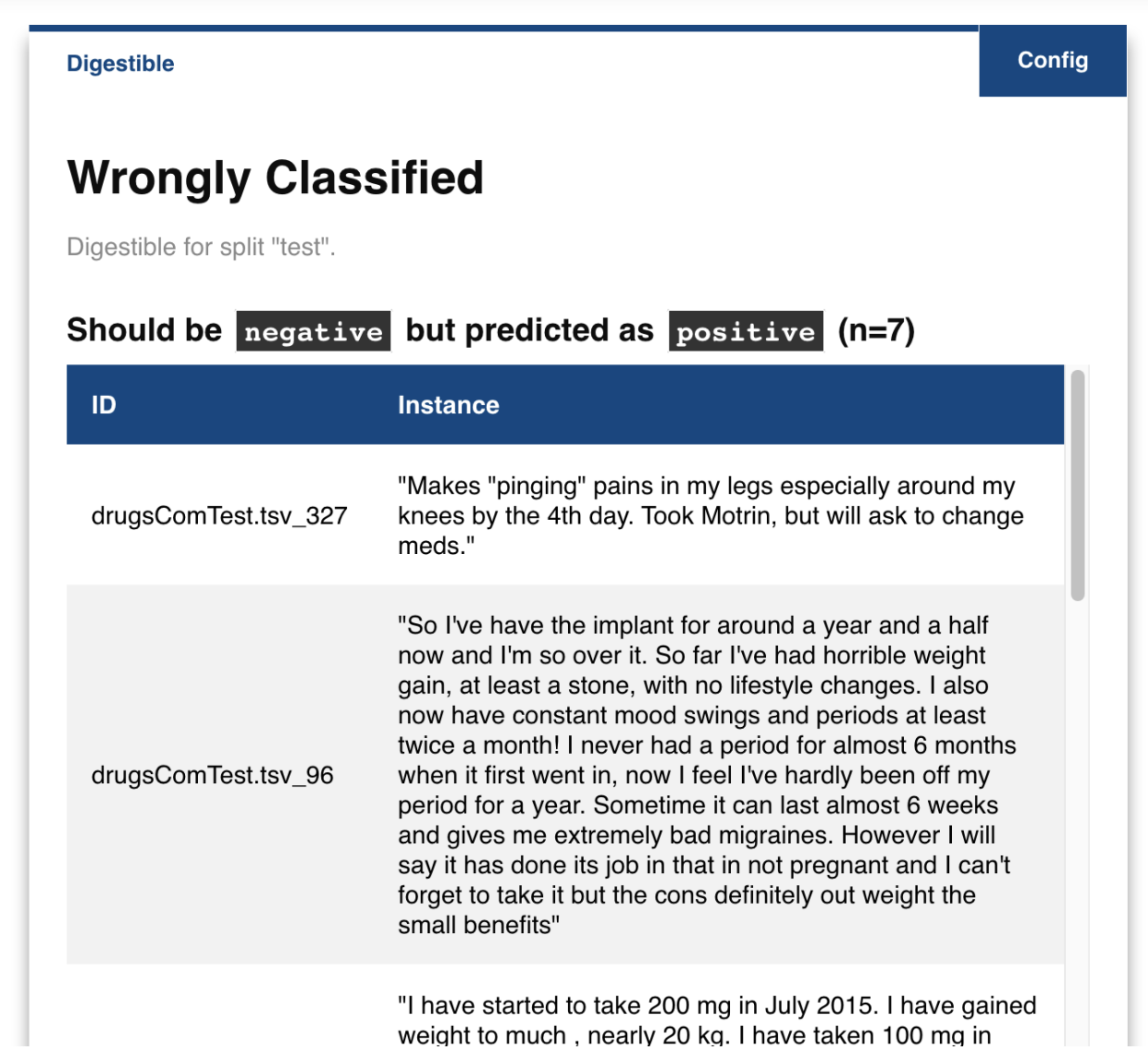
- Accuracy
- Precision
- Recall
- F1-score

That's some magic!

It sure is! The `explabox` inferred your model is a classifier, got all the dataset splits and did all the work for you. It even includes links to explain what all the metrics included mean! Some magic box, right?

The `explabox` even allows us to dive deeper into where the model went wrong. Let us see which examples were wrongly classified:

```
box.examine.wrongly_classified()
```



4. Explain

So what makes the `explabox` so unique? Why not use one of the many other tools for generating dataset descriptives and calculating performance?

Well, the `explabox` doesn't stop there! That's just where it starts. Ever heard of *explainable artificial intelligence* (XAI)? We've included that for you!

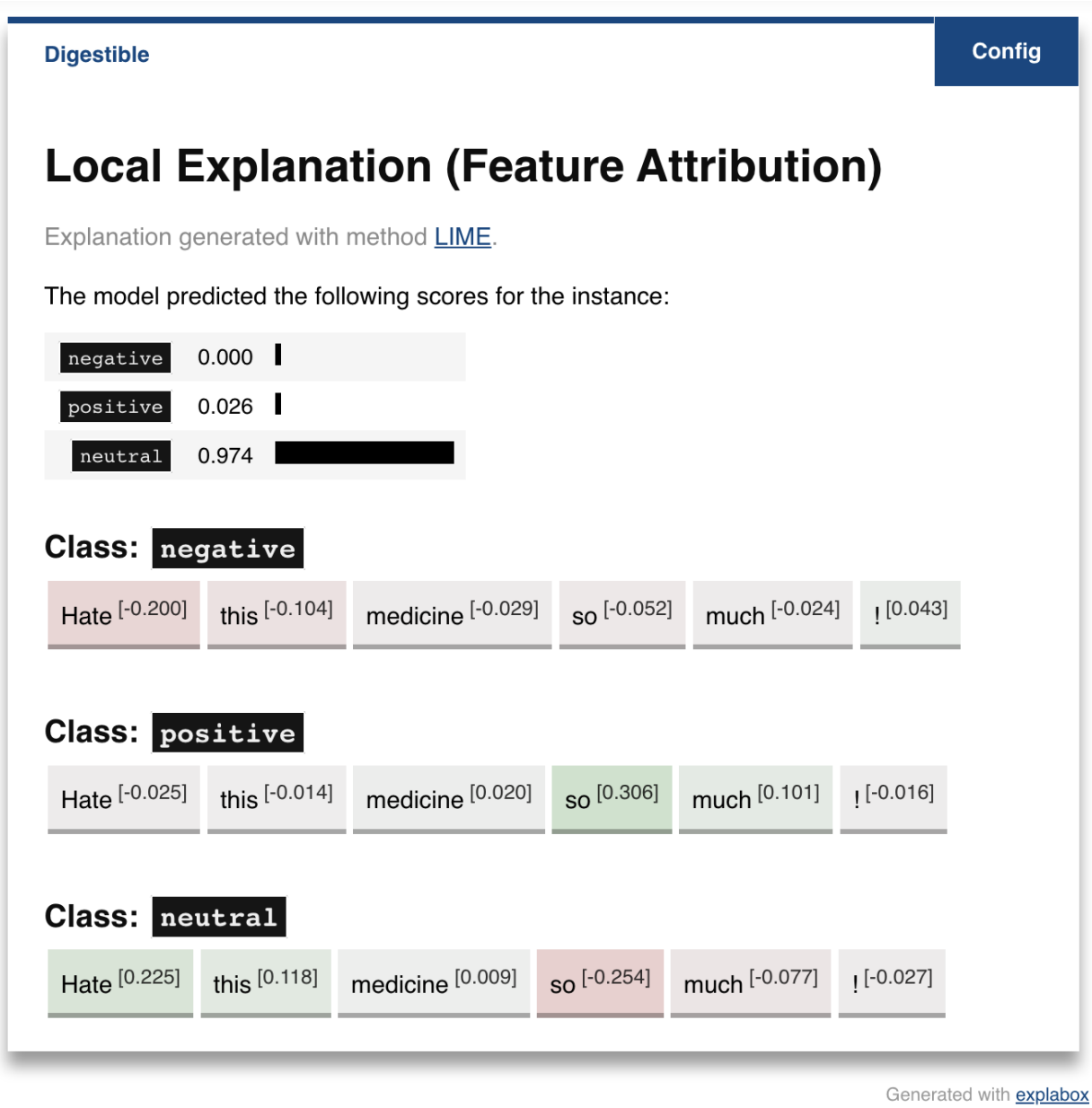
It doesn't matter if you use the explanations for yourself, show your end-user why a decision was made, to test an externally acquired model, or to provide model clients and supervisory authorities with the insights they require. We can help you on all of those. The explanations included are either *local* (providing explanations for a single prediction) or *global* (providing explanations for one or more dataset splits).

The *Explainer* creates explanations corresponding to a model and dataset (with ground-truth labels). The *Explainer* requires 'data' and 'model' defined. It is included in the `explabox` under the `.explain` property.

4.1 Local explanations

Why did my model predict a class label? Look no further than `box.explain.explain_prediction(...)`:

```
box.explain.explain_prediction('Hate this medicine so much!')
```



Generated with explabox

Even more magic!

We've got the work covered for you. Even though it is very easy to use the defaults, you can incorporate your own requirements into the function call:

- Want more samples? `.explain_prediction(..., n_instances=500)`
- Unweighed samples? `.explain_prediction(..., weigh_samples=False)`
- Want other methods? `.explain_prediction(..., methods=['lime', 'shap', 'local_tree'])`

So many options to choose from! It uses text_explainability for all these methods, which provides a generic architecture for constructing local/global explanation methods.

INFO

Want to see all options for local explanation? Check out the text_explainability documentation.

4.2 Global explanations

A lot of model behavior can be explained through the data it trained on. So, are there specific tokens corresponding to each label in the training set?

```
box.explain.token_frequency(splits='train', explain_model=False, labelwise=True)
```

We could have done the same for the `'test'` split, or explaining model predictions rather than ground-truth labels (`explain_model=True`) or aggregating them over all labels (`labelwise=False`). Want to know how informative tokens are in splitting labels? Try `box.explain.token_information(...)`.

The datasets include a lot of examples. Can we summarize them in fewer examples, let's say 5? That's what `box.explain.prototypes(...)` does:

```
box.explain.prototypes(n=5, method='kmedoids')
```

Or maybe add some outliers (so-called *criticisms*), that are a-typical for the dataset split:

```
box.explain.prototypes_criticisms(n_prototypes=5, n_criticisms=3)
```

INFO

There are so many options for explanations, provided for text datasets by the text_explainability package. Check it out to see what is possible!

5. Expose

Last, but far from least, the *Exposer* exposes your model and/or data, by performing sensitivity tests. With the *Exposer* you can see model sensitivity to random inputs (*safety*), test model generalizability (*robustness*), and see the effect of adjustments of attributes in the inputs (e.g. swapping male pronouns for female pronouns; *fairness*), for the dataset as a whole (*global*) as well as for individual instances (*local*).

The *Exposer* requires 'data' and 'model' defined. It is included in the `explabox` under the `.expose` property.

5.1 Safety

Does your text classifier break down at some inputs? Strings it cannot parse? Instances that empty or are too long? Try exposing the input space to see its safety:

```
box.expose.input_space('all', min_length=0, max_length=6000)
```

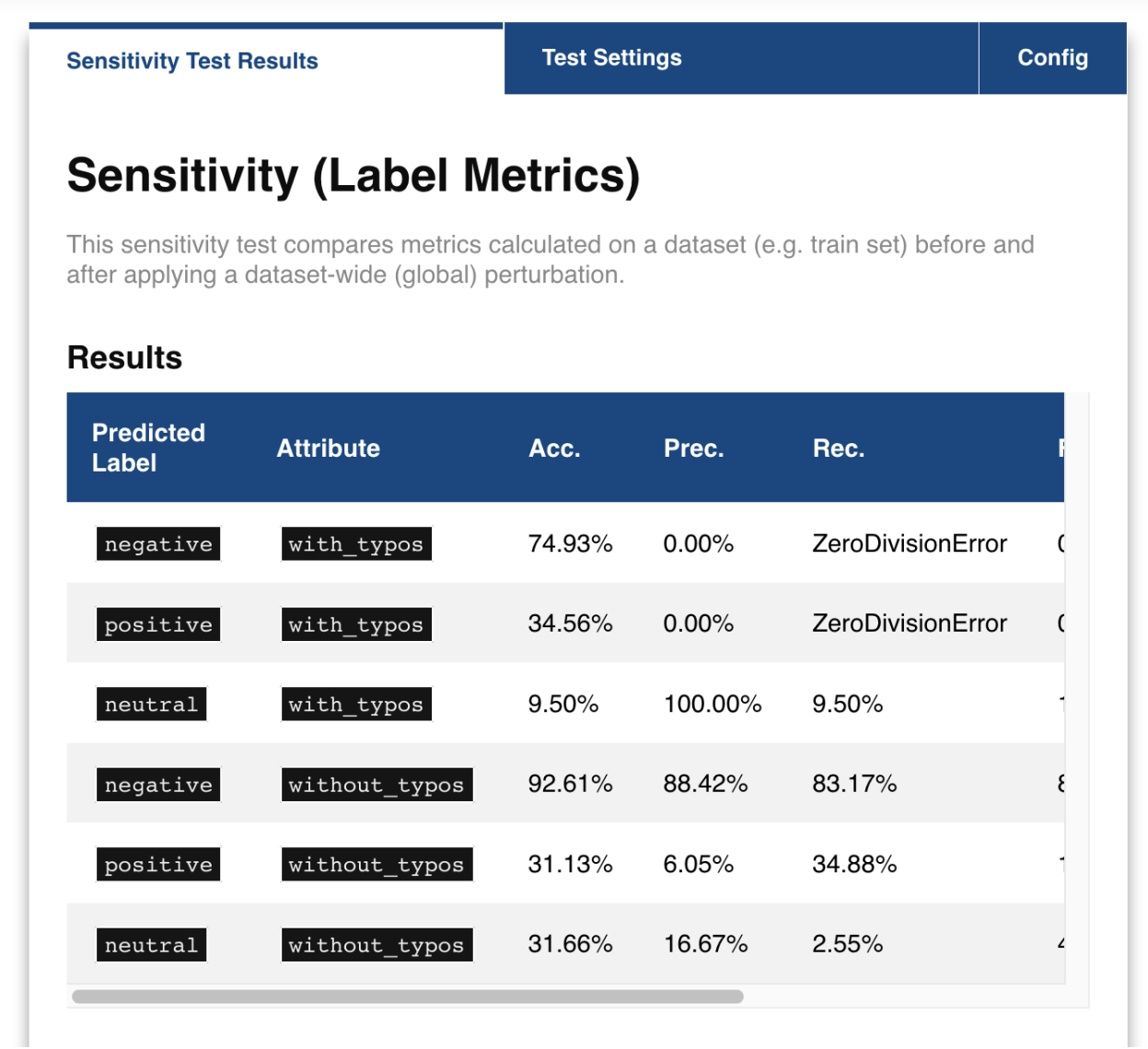
5.2 Robustness

Or take a global approach by seeing what happens if you transform all instances in the 'test' split from their original form to uppercase:

```
box.expose.compare_metric(perturbation='upper')
```

For both functions there are many techniques to choose from. Why not try exposing the input space with only `'ascii_upper'` and `'whitespace'`? Or try see how introducing `'add_typos'` affects your model?

```
box.expose.compare_metric(perturbation='add_typos')
```



5.2 Fairness & Robustness: pattern effects

Sometimes you need to go beyond the data to see model robustness and fairness. For the text domain, you can generate data with the text_sensitivity package and see how the models performs on them.

To do so, you write so-called *patterns* that generate data for you. At spots where you want some data filled in, you simply include curly braces and we fill in the data for you. For some entities (`name`, `city`, `email`, `year`, `month`, ...) we can even generate the data for you. Patterns with a pipe (`|`) simply put in the values you provided. Under the hood, it uses `from_pattern(...)` in the text_sensitivity example usage package. Example patterns include:

- Pattern `from_pattern('My phone number is {phone_number}')` generates *'My phone number is 001-364-187-2809', 'My phone number is +1-099-759-8699', ...*
- Pattern `from_pattern('{upper:name} is from {city}.)'` generates *'JAMES RUSSEL is from Oklahoma City.', 'BRIAN WILSON is from Millermouth.', ...*
- Pattern `from_pattern('{His|Her} favorite girl is {female_name}', female_name=RandomFirstName(sex='female'))` generates *'His favorite girl is Julia', 'Her favorite*

girl is Julia', ...

Let's turn that generated data into a proper test, where we except that the review is `positive` regardless:

```
box.expose.invariance('My friend {name} {loves|likes} this medicine. It is amazing!',
                      expectation='positive',
                      n=10)
```

Or one where it is `negative` regardless:

```
box.expose.invariance('My friend {upper:first_name} {hates|dislikes} this medicine. It is
↪{| not} terrible!',
                      expectation='negative',
                      n=2)
```

Or simply output the mean probabilistic score for the label `negative` for the generated instances:

```
box.expose.mean_score('My friend {first_name} from {city} {hates|dislikes} this medicine!
↪',
                      selected_labels='negative')
```

NOTE

For the text domain, sensitivity tests are provided by the text_sensitivity package. Try the text_sensitivity example usage guide to get acquainted with all that is possible.

6. Challenges

Want some pointers on where to go to next? Want to further `.explore`, `.examine`, `.expose` and `.explain` the black-box we provided? We've got some fun ideas to try out for yourself! Be sure to use the API Reference to figure out how to do them.

A. Which tokens are the most informative in distinguishing between the predicted classes?

B. Can I globally change the language to Dutch ('nl') if my data is Dutch?

C. How do local explanations with LIME compare against scores with KernelSHAP?

D. What happens if you change the `l1 regularization` of KernelSHAP?

E. How does the model perform if you repeat each sentence in the test set twice?

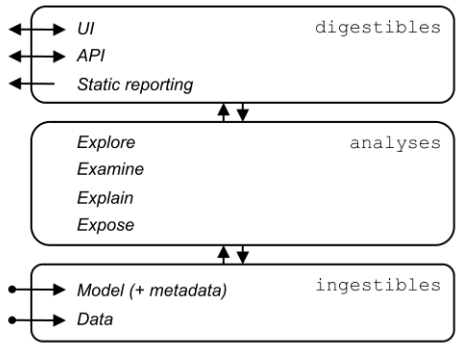
F. How does the model perform if you start each review with 'This is a review.'?

G. Does adding random typos degrade model performance?

H. Are there any drug names (https://www.drugs.com/drug_information.html) that seem to have more positive scores?

4.3 Overview

The Explabox aims to provide insights into your data and model behavior, by transforming *ingestibles* into *digestibles* through four types of *analyses*. The Explabox is split into three layers:



4.3.1 Ingestibles

Ingestibles encompass your `model` and `data`. The `Ingestible` class provides a unified interface for accessing your `model` (e.g. its meta-information and prediction function) and your `data` (e.g. the instances, the various splits, and corresponding ground-truth labels). Ingestibles contain meta-information of the data and model (to determine relevant functions for analyses and for auditability) and allow for optimized inferencing through batching and lazy loading.

Importing ingestibles

The `model` can be imported via the `import_model()` function, while the `data` is imported via the `import_data()` function.

Supported `models` are:

- *Scikit-learn*
- ONNX (supports *TensorFlow*, *PyTorch*, *HuggingFace*, *Scikit-Learn*, ...)
- Python callables

Supported `data` are:

- *NumPy* arrays
- *Pandas* DataFrames
- *HuggingFace* datasets
- Files (online or offline):
  - Tabular dataset files (`.csv`, `.tsv`, `.json`, `.txt`, `.xls`, `.xlsx`, ...)
  - Folders or archived files (`.zip`, `.rar`, `.gzip`, ...) containing multiple dataset files

  - Pickled Python objects (`.pkl`)
  - *HDF5* files

4.3.2 Analyses: turning ingestibles into digestibles

Once imported, these ingestibles can be made more informative by turning them into digestibles. Each analysis provides functions to enhance the transparency of the model and/or data:

Analysis	Class	Description	Requires
`explore`	`Explorer`	The Explorer explores your data by providing descriptive statistics.	*data*
`examine`	`Examiner`	The Examiner calculates quantitative metrics on how the model performs.	*data*, *model*
`explain`	`Explainer`	The Explainer creates explanations corresponding to a model and dataset (with ground-truth labels).	*data*, *model*
`expose`	`Exposer`	The Exposer exposes your model and/or data, by performing sensitivity tests.	*data*, *model*

The `Explabox` class is a unified interface for all of these classes, where an instance of each of these classes is constructed. These can be accessed via `.explore`, `.examine`, `.explain` and `.expose`, respectively.

For example, using the `Examiner` one can obtain (for a classification task) all instances wrongly classified by the model, which returns the `WronglyClassified` digestible:

```
>>> from explabox.examine import Examiner
>>> Examiner(ingestibles=ingestible).wrongly_classified()
```

4.3.3 Digestibles

Digestibles are the return type after performing an analysis. The information contained in them can be accessed in various ways (i.e. *interactively* or *statically*), depending on stakeholders needs. Example methods of access include:

- Python object (i.e. the raw property values)
- Object descriptions:
  - `JSON`
  - `YAML`
  - Python dictionary
- User interfaces (using plots rendered with `plotly`):
  - HTML
  - Jupyter Notebook

When using an interactive Python shell, the Explabox will default to the Jupyter Notebook interface. In a non-interactive Python shell the Python object is the basis for the digestible. For online applications (e.g. `streamlit`) the HTML interface can be used.

4.4 explabox

The Explabox aims to support data scientists and machine learning (ML) engineers in explaining, testing and documenting AI/ML models, developed in-house or acquired externally. The explabox turns your ingestibles (AI/ML model and/or dataset) into digestibles (statistics, explanations or sensitivity insights)!

To install run:

$ pip3 install explabox

Currently, the main interface for working with the Explabox is Jupyter Notebook. For more help, read the documentation at https://explabox.rtfd.io.

Explabox is developed by the Dutch National Police Lab AI (NPAI), and released under the GNU Lesser General Public License v3.0 (GNU LGPLv3).

class explabox.**Explabox**(*ingestibles=None*, *locale='en'*, ***kwargs*)

Bases: Readable, *IngestiblesMixin*

Use the Explabox to *.explore*, *.examine*, *.expose* and *.explain* your AI model.

Example

```
>>> from explabox import Explabox
>>> box = Explabox(data=data, model=model)
```

Parameters

- **ingestibles** (*Optional[Ingestible], optional*) – Ingestibles (data and model). Defaults to None.
- **locale** (*str, optional*) – Language of dataset. Defaults to 'en'.
- ****kwargs** – Arguments used to construct an Ingestible (if the ingestibles argument is None).

Subpackages:

4.4.1 explabox.digestibles

Ingestibles are turned into digestibles, containing information to explore/examine/explain/expose your model.

class explabox.digestibles.**Dataset**(*instances*, *labels*, *type='dataset'*, *subtype=None*, *callargs=None*, ***kwargs*)

Bases: MetaInfo

Digestible for dataset.

Examples

Construct a dataset with 5 instances and get instance 2 through 4:

```
>>> dataset = Dataset(instances, ['positive', 'negative', 'positive', 'neutral',
↪'positive'])
>>> dataset[2:4]
```

Get the first instance:

```
>>> dataset.head(n=1)
```

Randomly sample two instances:

```
>>> dataset.sample(n=2, seed=0)
```

Get all instances in the dataset labelled as 'positive':

```
>>> dataset.filter('positive')
```

Parameters

- **instances** (*_type_*) – Instances.
- **labels** (*Sequence[LT]*) – Ground-truth labels (annotated).
- **type** (*str, optional*) – Type description. Defaults to "dataset".
- **subtype** (*Optional[str], optional*) – Subtype description. Defaults to None.
- **callargs** (*Optional[dict], optional*) – Call arguments for reproducibility. Defaults to None.

property content

Content as dictionary.

property data

Get data property.

filter(*indexer*)

Filter dataset by label, filter function or boolean list/array.

Examples

Filter by label 'positive':

```
>>> dataset.filter('positive')
```

Filter if '@' character in data:

```
>>> dataset.filter(lambda data, label: '@' in data)
```

Filter if '@' character not in instance and label in ('neutral', 'negative'):

```
>>> def filter_fn(instance):
...     return '@' not in instance['data'] and instance['label'] in (frozenset({
↪'neutral'}), frozenset({'negative'}))
>>> dataset.filter(filter_fn)
```

Filter by boolean sequence (should be equal length to the number of instances):

```
>>> dataset.filter([True] * len(dataset))
```

Parameters
indexer (*Union[Callable[[dict], bool], Callable[[DT, LT], bool], Sequence[bool], LT]*) – Filter to apply.

Raises
ValueError – Boolean array should be equal length to number of instances.

Returns
Filtered dataset.

Return type
Dataset

get_by_index(*index*)

Get item(s) by integer index.

Return type
Dataset

get_by_key(*index*)

Get item(s) by key.

Return type
Dataset

head(*n=10*)

Get the first n elements in the dataset.

Parameters
n (*int, optional*) – Number of elements >= 0. Defaults to 10.

Raises
ValueError – n should be >= 0.

Returns
First n elements.

Return type
Dataset

property instances

Get instances property

property keys

Get keys property

property labels

Get labels property.

sample(*n=1*, *seed=None*)

Get a random sample of size n.

Parameters

- **n** (*int, optional*) – Number of elements >= 0. Defaults to 1.
- **seed** (*int, optional*) – Seed for reproducibility; if None it takes a random seed. Defaults to None.

Raises

ValueError – n should be >= 0.

Returns

Random subsample.

Return type

Dataset

tail(*n=10*)

Get the last n elements in the dataset.

Parameters

n (*int, optional*) – Number of elements >= 0. Defaults to 10.

Raises

ValueError – n should be >= 0.

Returns

Last n elements.

Return type

Dataset

class explabox.digestibles.**Descriptives**(*labels*, *label_counts*, *tokenized_lengths*, *type='descriptives'*, *callargs=None*, ***kwargs*)

Bases: MetaInfo

Digestible for descriptive statistics.

Parameters

- **labels** (*Sequence[LT]*) – Names of labels.
- **label_counts** (*Dict[str, Dict[LT, int]]*) – Counts per label per split.
- **tokenized_lengths** (*dict*) – Descriptive statistics for lengths of tokenized instances.
- **type** (*str, optional*) – Type description. Defaults to "descriptives".
- **subtype** (*Optional[str], optional*) – Subtype description. Defaults to None.
- **callargs** (*Optional[dict], optional*) – Call arguments for reproducibility. Defaults to None.

property content

Content as dictionary.

class explabox.digestibles.**Performance**(*labels*, *metrics*, *type='model_performance'*, *subtype='classification'*, *callargs=None*, ***kwargs*)

Bases: MetaInfo

Digestible for performance metrics.

Parameters

- **labels** (*Sequence[LT]*) – Names of labels.
- **metrics** (*dict*) – Performance metrics per label.
- **type** (*str, optional*) – Type description. Defaults to "model_performance".
- **subtype** (*Optional[str], optional*) – Subtype description. Defaults to None.
- **callargs** (*Optional[dict], optional*) – Call arguments for reproducibility. Defaults to None.

property content

Content as dictionary.

property metrics

Metrics values.

class explabox.digestibles.**WronglyClassified**(*instances*, *contingency_table*, *type='wrongly_classified'*, *callargs=None*, ***kwargs*)

Bases: Instances

Digestible for wrongly classified instances

Parameters

- **instances** (*_type_*) – Instances.
- **contingency_table** (*Dict[Tuple[LT, LT], FrozenSet[KT]]*) – Classification contingency table as returned from *instancelib.analysis.base.contingency_table()*.
- **type** (*str, optional*) – Type description. Defaults to "wrongly_classified".
- **callargs** (*Optional[dict], optional*) – Call arguments for reproducibility. Defaults to None.

property content

Content as dictionary.

property wrongly_classified

Wrongly classified instances, grouped by their ground-truth value, predicted value and instances.

4.4.2 explabox.examine

Calculate quantitative metrics on how the model performs, and examine where the model went wrong.

class explabox.examine.**Examiner**(*data=None*, *model=None*, *ingestibles=None*, ***kwargs*)

Bases: Readable, *ModelMixin*, *IngestiblesMixin*

The Examiner calculates quantitative metrics on how the model performs.

The Examiner requires 'data' and 'model' defined. It is included in the Explabox under the *.examine* property.

Examples

Construct the examiner:

```
>>> from explabox.examine import Examiner
>>> examiner = Explainer(data=data, model=model)
```

Calculate model performance metrics on the validation set:

```
>>> examiner(split='validation')
```

See all wrongly classified examples in the test set:

```
>>> examiner.wrongly_classified(split='test')
```

Parameters

- **data** (*Optional[Environment], optional*) – Data for ingestibles. Defaults to None.
- **model** (*Optional[AbstractClassifier], optional*) – Model for ingestibles. Defaults to None.
- **ingestibles** (*Optional[Ingestible], optional*) – Ingestible. Defaults to None.

performance(*split='test'*, ***kwargs*)

Determine performance metrics, the amount of predictions for each label in the test set and the values for the confusion matrix for each label in the test set.

Parameters
split (*str, optional*) – Split to calculate metrics on. Defaults to 'test'.

Returns
Performance metrics of your model on the split.

Return type
Performance

wrongly_classified(*split='test'*, ***kwargs*)

Give all wrongly classified samples.

Parameters
split (*str, optional*) – Name of split. Defaults to 'test'.

Returns
Wrongly classified examples in this split.

Return type
WronglyClassified

4.4.3 explabox.explain

Add explainability to your model/dataset with the Explainer class.

Subpackages:

explabox.explain.text

Add explainability to your text model/dataset.

class explabox.explain.text.**Explainer**(*data=None*, *model=None*, *ingestibles=None*, ***kwargs*)

Bases: Readable, *IngestiblesMixin*

The Explainer creates explanations corresponding to a model and dataset (with ground-truth labels).

With the Explainer you can use explainble AI (XAI) methods for explaining the whole dataset (global), model behavior on the dataset (global), and specific predictions/decisions (local).

The Explainer requires 'data' and 'model' defined. It is included in the Explabox under the *.explain* property.

Examples

Construct the explainer:

```
>>> from explabox.explain import Explainer
>>> explainer = Explainer(data=data, model=model)
```

Get a local explanation with LIME (https://github.com/marcotcr/lime) and kernelSHAP (https://github.com/slundberg/shap):

```
>>> explainer.explain_prediction('I love this so much!', methods=['lime', 'kernel_
↪shap'])
```

See the top-25 tokens for predicted classifier labels on the test set:

```
>>> explainer.token_frequency(k=25, explain_model=True, splits='test')
```

Select the top-5 prototypical examples in the train set:

```
>>> explainer.prototypes(n=5, splits='train')
```

Parameters

- **data** (*Optional[Environment], optional*) – Data for ingestibles. Defaults to None.
- **model** (*Optional[AbstractClassifier], optional*) – Model for ingestibles. Defaults to None.
- **ingestibles** (*Optional[Ingestible], optional*) – Ingestible. Defaults to None.

explain_prediction(*sample*, **args*, *methods=['lime']*, ***kwargs*)

Explain specific sample locally.

Parameters

- **sample** (Union[int, str]) – Identifier of sample in dataset (int) or input (str).

- **methods** (Union[str, List[str]]) – List of methods to get explanations from. Choose from 'lime', 'shap', 'baylime', 'tree', 'rules', 'foil_tree'.
- ***args** – Positional arguments passed to local explanation technique.
- ****kwargs** – Keyword arguments passed to local explanation technique.

Returns
Explanations for each selected method, unless method is unknown (returns None).

Return type
Optional[*MultipleReturn*]

prototypes(*method='mmdcritic'*, *n=5*, *splits='test'*, *embedder=<class 'text_explainability.data.embedding.TfidfVectorizer'>*, *labelwise=False*, *seed=0*)

Select n prototypes (representative samples) for the given split(s).

Parameters

- **method** (*str, optional*) – Method(s) to apply. Choose from ['mmdcritic', 'kmedoids']. Defaults to 'mmdcritic'.
- **n** (*int, optional*) – Number of prototypes to generate. Defaults to 5.
- **splits** (*Union[str, List[str]], optional*) – Name(s) of split(s). Defaults to "test".
- **embedder** (*Optional[Embedder], optional*) – Embedder used. Defaults to TfidfVectorizer.
- **labelwise** (*bool, optional*) – Select for each label. Defaults to False.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

Raises
ValueError – Unknown method selected.

Returns
Prototypes for each methods and split.

Return type
Union[Instances, *MultipleReturn*]

prototypes_criticisms(*n_prototypes=5*, *n_criticisms=3*, *splits='test'*, *embedder=<class 'text_explainability.data.embedding.TfidfVectorizer'>*, *labelwise=False*, ***kwargs*)

Select n prototypes (representative samples) and n criticisms (outliers) for the given split(s).

Parameters

- **n_prototypes** (*int, optional*) – Number of prototypes to generate. Defaults to 5.
- **n_criticsms** (*int, optional*) – Number of criticisms to generate. Defaults to 3.
- **splits** (*Union[str, List[str]], optional*) – Name(s) of split(s). Defaults to "test".
- **embedder** (*Optional[Embedder], optional*) – Embedder used. Defaults to TfidfVectorizer.
- **labelwise** (*bool, optional*) – Select for each label. Defaults to False.
- **n_criticisms** (*int*) –

Returns
Prototypes for each methods and split.

Return type
Union[Instances, *MultipleReturn*]

token_frequency(*splits='test'*, *explain_model=True*, *labelwise=True*, *k=25*, *filter_words=<Proxy at 0x7fedfd57c540 wrapping ['de', 'het', 'een'] at 0x7fedfd1c1440 with factory <function lazy.<locals>.<lambda>>>*, *lower=True*, *seed=0*, ***count_vectorizer_kwargs*)

Show the top-k number of tokens for each ground-truth or predicted label.

Parameters

- **splits** (*Union[str, List[str]], optional*) – Split names to get the explanation for. Defaults to 'test'.
- **explain_model** (*bool, optional*) – Whether to explain the model (True) or ground-truth labels (False). Defaults to True.
- **labelwise** (*bool, optional*) – Whether to summarize the counts for each label seperately. Defaults to True.
- **k** (*Optional[int], optional*) – Limit to the top-k words per label, or all words if None. Defaults to 25.
- **filter_words** (*List[str], optional*) – Words to filter out from top-k. Defaults to ['a', 'an', 'the'].
- **lower** (*bool, optional*) – Whether to make all tokens lowercase. Defaults to True.
- **seed** (*int, optional*) –
- ****count_vectorizer_kwargs** – Optional arguments passed to *CountVectorizer*/*FastCountVectorizer*.

Returns
Each label with corresponding top words and their frequency

Return type
Union[FeatureList, *MultipleReturn*]

token_information(*splits='test'*, *explain_model=True*, *k=25*, *filter_words=<Proxy at 0x7fedfd589700 wrapping ['de', 'het', 'een'] at 0x7fedfd1d2900 with factory <function lazy.<locals>.<lambda>>>*, *lower=True*, *seed=0*, ***count_vectorizer_kwargs*)

Show the top-k token mutual information for a dataset or model.

Parameters

- **splits** (*Union[str, List[str]], optional*) – Split names to get the explanation for. Defaults to 'test'.
- **explain_model** (*bool, optional*) – Whether to explain the model (True) or ground-truth labels (False). Defaults to True.
- **labelwise** (*bool, optional*) – Whether to summarize the counts for each label seperately. Defaults to True.
- **k** (*Optional[int], optional*) – Limit to the top-k words per label, or all words if None. Defaults to 25.
- **filter_words** (*List[str], optional*) – Words to filter out from top-k. Defaults to ['a', 'an', 'the'].
- **lower** (*bool, optional*) – Whether to make all tokens lowercase. Defaults to True.

- **seed** (*int, optional*) –
- ****count_vectorizer_kwargs** – Optional arguments passed to *CountVectorizer*/*FastCountVectorizer*.

Returns

k labels, sorted based on their mutual information with
the output (predictive model labels or ground-truth labels)

Return type
Union[FeatureList, *MultipleReturn*]

4.4.4 explabox.explore

Functions/classes for exploring your data (dataset descriptives).

class explabox.explore.**Explorer**(*data=None*, *ingestibles=None*, ***kwargs*)

Bases: Readable, *IngestiblesMixin*

The Explorer explores your data by providing descriptive statistics.

The Explorer requires 'data' defined. It is included in the Explabox under the *.explore* property.

Examples

Get dataset descriptives:

```
>>> from explabox.explore import Explorer
>>> explorer = Explorer(data=data)
>>> explorer()
```

Show the first 10 instances of the test split

```
>>> from explabox.explore import Explorer
>>> explorer = Explorer(data=data)
>>> explorer.instances(split="test")[:10]
```

Parameters

- **data** (*Optional[Environment], optional*) – Data for ingestibles. Defaults to None.
- **ingestibles** (*Optional[Ingestible], optional*) – Ingestible. Defaults to None.

descriptives(***kwargs*)

Describe features such as the amount per label for the train, test and model predictions and text data specific features such as the maximum/minimum/mean amount of words in a sample and the standard deviation.

Returns
Descriptive statistics of each split.

Return type
Descriptives

instances(*split='test'*, ***kwargs*)

Get the instances of the given split.

Parameters
split (*str, optional*) – Split to select. Defaults to "test".

Returns
Instances in the split.

Return type
Dataset

4.4.5 explabox.expose

Functions/classes for sensitivity testing (fairness and robustness).

Subpackages:

explabox.expose.text

Functions/classes for sensitivity testing (fairness and robustness) for text data.

class explabox.expose.text.**Exposer**(*data=None*, *model=None*, *ingestibles=None*, ***kwargs*)

Bases: Readable, *IngestiblesMixin*

The Exposer exposes your model and/or data, by performing sensitivity tests.

With the Exposer you can see model sensitivity to random inputs (robustness), test model generalizability (robustness), and see the effect of adjustments of attributes in the inputs (e.g. swapping male pronouns for female pronouns; fairness), for the dataset as a whole (global) as well as for individual instances (local).

The Exposer requires 'data' and 'model' defined. It is included in the Explabox under the *.expose* property.

Examples

See how performance of a model on the test dataset is affected when text is randomly changed to uppercase:

```
>>> from explabox.expose import Exposer
>>> exposer = Exposer(data=data, model=model)
>>> exposer.compare_metric(splits='test', perturbation='random_upper')
```

Parameters

- **data** (*Optional[Environment], optional*) – Data for ingestibles. Defaults to None.
- **model** (*Optional[AbstractClassifier], optional*) – Model for ingestibles. Defaults to None.
- **ingestibles** (*Optional[Ingestible], optional*) – Ingestible. Defaults to None.

compare_metric(*perturbation*, *splits='test'*)

Compare metrics for each ground-truth label and attribute after applying a dataset-wide perturbation.

Examples

Compare metric of model performance (e.g. accuracy, precision) before and after mapping each instance in the test dataset to uppercase:

```
>>> box.expose.compare_metric(splits='test', peturbation='upper')
```

Add '!!!' to the end of each text in the 'train' and 'test' split and see how it affects performance:

```
>>> from explabox.expose.text import OneToOnePerturbation
>>> perturbation_fn = OneToOnePerturbation(lambda x: f'{x}!!!')
>>> box.expose.compare_metrics(splits=['train', 'test'],␣
↪perturbation=perturbation_fn)
```

Parameters

- **perturbation** (*Union[OneToOnePerturbation, str]*) – Custom perturbation or one of the default ones, picked by their string: 'lower', 'upper', 'random_lower', 'random_upper', 'add_typos', 'random_case_swap', 'swap_random' (swap characters), 'delete_random' (delete characters), 'repeat' (repeats twice).
- **splits** (*Union[str, List[str]], optional*) – Split to apply the perturbation to. Defaults to "test".

Raises

ValueError – Unknown perturbation.

Returns

Original label (before perturbation), perturbed label (after perturbation) and metrics for label-attribute pair.

Return type

Union[LabelMetrics, *MultipleReturn*]

input_space(*generators*, *n_samples=100*, *min_length=0*, *max_length=100*, *seed=0*, ***kwargs*)

Test the robustness of a machine learning model to different input types (safety).

Example

Test a pretrained black-box *model* for its robustness to 1000 random strings (length 0 to 500), containing whitespace characters, ASCII (upper, lower and numbers), emojis and Russian Cyrillic characters:

```
>>> from explabox import Explabox, RandomEmojis, RandomCyrillic
>>> box = Explabox(data=data, model=model)
>>> box.expose.input_space(generators=['whitespace', 'ascii',␣
↪RandomEmojis(base=True), RandomCyrillic('ru')],
...                        n_samples=1000,
...                        min_length=0,
...                        max_length=500)
```

Parameters

- **generators** (*Union[str, RandomString, List[Union[RandomString, str]]]*) – Random character generators. If 'all' select all generators. For strings

choose from 'ascii', 'emojis, 'whitespace', 'spaces', 'ascii_upper', 'ascii_lower', 'digits', 'punctuation', 'cyrillic'.

- **n_samples** (*int, optional*) – Number of test samples. Defaults to 100.
- **min_length** (*int, optional*) – Input minimum length. Defaults to 0.
- **max_length** (*int, optional*) – Input maximum length. Defaults to 100.
- **seed** (*Optional[int], optional*) – Seed for reproducibility purposes. Defaults to 0.

Returns
Percentage of success cases, list of succeeded/failed instances

Return type
SuccessTest

invariance(*pattern*, *expectation*, ***kwargs*)

Test for the failure rate under invariance.

Example

Test if predictions remain 'positive' for 50 samples of the pattern *'I {like|love} {name} from {city}!'*:

```
>>> from explabox import Explabox
>>> box = Explabox(data=data, model=model)
>>> box.expose.invariance('I {like|love} {name} from {city}!', expectation=
↪'positive', n_samples=50)
```

Parameters

- **pattern** (*str*) – String pattern to generate examples from.
- **expectation** (*Optional[LT], optional*) – Expected outcome values. Defaults to None.
- ****kwargs** – Optional arguments passed onto the *data.generate.from_pattern()* function.

Returns
Percentage of success cases, list of succeeded (invariant)/failed (variant) instances

Return type
SuccessTest

mean_score(*pattern*, *selected_labels='all'*, ***kwargs*)

Calculate mean (probability) score for the given labels, for data generated from a pattern.

Example

Calculate the mean score for the 'positive' label for *'I {like|love} {name} from {city}!'*:

```
>>> from explabox import Explabox
>>> box = Explabox(data=data, model=model)
>>> box.expose.mean_score('I {like|love} {name} from {city}!', selected_labels=
↪'positive', seed=0)
```

Parameters

- **pattern** (*str*) – Pattern to generate instance from.
- **selected_labels** (*Optional[Union[LT, List[LT]]], optional*) – Label name to select(s). If None or 'all' it is replaced by all labels. Defaults to 'all'.
- ****kwargs** – Optional arguments passed onto the *data.generate.from_pattern()* function.

Return type
MeanScore | MultipleReturn

Returns
Mean score for one label or all selected labels.

Return type
Union[MeanScore, *MultipleReturn*]

Parameters

- **pattern** (*str*) –
- **selected_labels** (*LT | List[LT] | None*) –

Submodules:

explabox.expose.text.characters module

Character-level perturbations.

explabox.expose.text.characters.**add_typos**(*n=1*, ***kwargs*)

Create a *Perturbation* object that adds keyboard typos within words.

Parameters

- **n** (*int, optional*) – Number of perturbed instances required. Defaults to 1.
- ****kwargs** – See **`naw.KeyboardAug`_** for optional constructor arguments.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**delete_random**(*n=1*, ***kwargs*)

Create a *Perturbation* object with random character deletions in words.

Parameters

- **n** (*int, optional*) – Number of perturbed instances required. Defaults to 1.
- ****kwargs** – See nac.RandomCharAug for optional constructor arguments (uses *action='delete'* by default).

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**random_case_swap**(*n=1*)

Create a *Perturbation* object that randomly swaps characters case (lower to higher or vice versa).

Parameters
n (*int, optional*) – Number of perturbed instances required. Defaults to 1.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**random_lower**(*n=1*)

Create a *Perturbation* object that randomly swaps characters to lowercase.

Parameters
n (*int, optional*) – Number of perturbed instances required. Defaults to 1.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**random_spaces**(*n=1*, ***kwargs*)

Create a *Perturbation* object that adds random spaces within words (splits them up).

Parameters

- **n** (*int, optional*) – Number of perturbed instances required. Defaults to 1.
- ****kwargs** – See naw.SplitAug for optional constructor arguments.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**random_upper**(*n=1*)

Create a *Perturbation* object that randomly swaps characters to uppercase.

Parameters
n (*int, optional*) – Number of perturbed instances required. Defaults to 1.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.characters.**swap_random**(*n=1*, ***kwargs*)

Create a *Perturbation* object that randomly swaps characters within words.

Parameters

- **n** (*int, optional*) – Number of perturbed instances required. Defaults to 1.
- ****kwargs** – See nac.RandomCharAug for optional constructor arguments (uses *action='swap'* by default).

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.sentences module

Sentence-level perturbations.

explabox.expose.text.sentences.**repeat_k_times**(*k=10*, *connector=' '*)

Repeat a string k times.

Parameters

- **k** (*int, optional*) – Number of times to repeat a string. Defaults to 10.
- **connector** (*Optional[str], optional*) – Connector between adjacent repeats. Defaults to ‘ ‘.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.sentences.**to_lower**()

Make all characters in a string lowercase.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.sentences.**to_upper**()

Make all characters in a string uppercase.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

explabox.expose.text.words module

Word-level perturbations.

4.4.6 explabox.ingestibles

Ingestibles are your model and data, which can be turned into digestibles that explore/examine/explain/expose your data and/or model.

class explabox.ingestibles.**Ingestible**(*data=None*, *model=None*, *splits={'test': 'test', 'train': 'train', 'validation': 'validation'}*)

Bases: dict

Parameters

- **data** (*Environment | None*) –

- **model** (*AbstractClassifier | None*) –
- **splits** (*Dict[KT, KT]*) –

check_requirements(*elements=['data', 'model']*)

Check if the required elements are in the ingestibles.

Parameters
elements (*List[str], optional*) – Elements to check. Defaults to [‘data’, ‘model’].

Raises
ValueError – The required element is not in the ingestibles.

Returns
True if all requirements are included.

Return type
bool

property data

get_named_split(*name*, *validate=False*)

Get split by name.

Parameters

- **name** (*KT*) – Name of split.
- **validate** (*bool, optional*) – Return None if no split is found or throw an error. Defaults to False.

Raises
ValueError – Unknown split

Returns
Provider of split if it exists, else None.

Return type
Optional[InstanceProvider]

property labels

Labelprovider.

property labelset

Label names.

property model

Predictive model.

property splits

Names of splits.

property test

Test data split.

property train

Train data split.

property validation

Validation data split.

Subpackages:

explabox.ingestibles.data

Handling of data.

explabox.ingestibles.model

Functions to import models from the genbase library.

4.4.7 explabox.ui

User interfaces.

Submodules:

explabox.ui.notebook module

User interface for Jupyter notebook.

`explabox.ui.notebook.MAIN_COLOR`

Default color for the notebook UI.

Type
str

`explabox.ui.notebook.PACKAGE_LINK`

URL to package.

Type
str

`explabox.ui.notebook.PACKAGE_NAME`

Name of package.

Type
str

`class explabox.ui.notebook.GBRenderRestyled(*configs)`

Bases: `Render`, *RestyleMixin*

Restyle the *genbase* renderer.

`class explabox.ui.notebook.Render(*configs)`

Bases: *GBRenderRestyled*

Custom renderer for *explabox*.

`format_title(title, h='h1', **renderargs)`

Format title in HTML format.

Parameters

- **title** (*str*) – Title contents.
- **h** (*str, optional*) – h-tag (h1, h2, ...). Defaults to 'h1'.

Returns
Formatted title.

Return type
str

`get_renderer`(*meta*)

Get a render function (Callable taking *meta*, *content* and ***renderargs* and returning a *str*).

Parameters
meta (*dict*) – Meta information to decide on appropriate renderer.

`render_subtitle`(*meta*, *content*, ***renderargs*)

Return type
str

Parameters

- **meta** (*dict*) –
- **content** (*dict*) –

class explabox.ui.notebook.**RestyleMixin**

Bases: object

Adds *self.restyle()* function to apply main_color and package_link.

`restyle`()

class explabox.ui.notebook.**TERenderRestyled**(**configs*)

Bases: Render, *RestyleMixin*

Restyle the *text_explainability* renderer.

class explabox.ui.notebook.**TSRenderRestyled**(**configs*)

Bases: Render, *RestyleMixin*

Restyle the *text_sensitivity* renderer.

explabox.ui.notebook.**dataset_renderer**(*meta*, *content*, ***renderargs*)

Renderer for *explabox.digestibles.Dataset*.

explabox.ui.notebook.**descriptives_renderer**(*meta*, *content*, ***renderargs*)

Renderer for *explabox.digestibles.Descriptives*.

explabox.ui.notebook.**format_table**(*header*, *content*)

Format a HTML table based on header and content.

explabox.ui.notebook.**replace_renderer**(*res*)

Replace a renderer from a function result with a restyled one.

explabox.ui.notebook.**restyle**(*function*)

Apply a decorator for restyling the returned renderer.

Parameters
function (*Callable*) –

explabox.ui.notebook.**wrongly_classified_renderer**(*meta*, *content*, ***renderargs*)

Rendered for *explabox.digestibles.WronglyClassified*.

4.4.8 explabox.utils

Utility functions and classes.

class explabox.utils.**MultipleReturn**(**return_values*)

Bases: object

Holds multiple return values (e.g. from *return_types*) in one iterable return value.

property html

property raw_html

to_config()

Submodules:

explabox.utils.io module

Utility functions for input/output behavior.

explabox.utils.io.**create_output_dir**(*path='/home/docs/checkouts/readthedocs.org/user_builds/explabox/checkouts/latest/doc*

Create the directory to write results to.

Parameters
path (*str, optional*) – Path of the directory where to write any results to. Defaults to OUTPUT_DIR.

Submodules:

4.4.9 explabox.config module

Configuration for default paths and variables.

4.4.10 explabox.mixins module

Extensions to classes.

class explabox.mixins.**IngestiblesMixin**

Bases: object

check_requirements(*elements=['data', 'model']*)

Check if the required elements are in the ingestibles.

Parameters
elements (*List[str], optional*) – Elements to check. Defaults to [‘data’, ‘model’].

Raises
ValueError – The required element is not in the ingestibles.

Returns
True if all requirements are included.

Return type
bool

property data
All data.

property labels
Labelprovider.

property labelset
Names of labels.

property model
Predictive model.

property splits
Named splits.

class explabox.mixins.**ModelMixin**
Bases: object

property is_classifier: bool
Whether the included model is a classifier (True) or not (False).

4.5 Changelog

All notable changes to explabox will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

4.5.1 Unreleased

Added

- Example usage in Google Colab (requires genbase>=0.2.17)
- explabox.explore.instances to summarize instances in a split
- Selecting, sampling and filtering of digestibles.Dataset
- Support and testing for Python 3.11

Fixed

- Support for new pandas versions
- Bugfix where tokens are not properly filtered in TokenFrequency (requires text-explainability>=0.6.6)

4.5.2 0.9b7

Added

- Makefile for Windows (`make.bat`)
- License to each file
- Coverage testing on `codecov.io`
- Hosting documentation on `readthedocs.io`
- Dependencies for `explabox[dev]`, `explabox[docs]` and `explabox[all]`
- Contribution guide
- Software testing on Python `3.8`, `3.9` and `3.10`; for `windows`, `ubuntu` and `macos`
- Issue templates
- Pull request template
- Example usage with `explabox-demo-drugreview`
- Finished all docstrings
- Return `html` and `raw_html` from MultipleReturn
- Probability scores to feature contributions

Changed

- Ported repository to GitHub
- CI/CD pipeline for GitHub actions

Fixed

- Ensured stable dependency versions of `instancelib` and `text_explainability`
- `numpy.str` deprecation warning

Removed

- Components from Gitlab repository

4.5.3 0.9b6

Added

- README.md
- Sphinx documentation
- Makefile
- License
- Requirements

- Dataset descriptives
- Installation guide
- Text sensitivity tests (`text_sensitivity`)
- Text explainability (`text_explainability`)
- Model importing (`genbase`)
- Dataset handling (`genbase`)
- Basic UI (`genbase`)
- `git` setup

4.6 Contributing

We would love for you to contribute and improve the `explabox`!

If you are unable to contribute yourself, feel free to post a feature request or bug report for the developers and maintainers at the Netherlands *National Police Lab AI* (NPAI).

4.6.1 1. Getting started

- Make sure you have a GitHub account
- Submit a ticket for your issue, assuming one does not already exist
- Fork the repository on GitHub

4.6.2 2. Setting up your environment

To ensure you are able to interactively edit and test your code, when contributing we recommend you install your forked version of the `explabox` with the `-e` (*editable*; i.e. `pip3 install -e .`) flag. Depending on where you want to contribute to, we have also provided you with the necessary optional packages required for quality checks and/or package building. These are:

- General development: `pip3 install -e ".[dev]"`
- Only working on documentation: `pip3 install -e ".[docs]"`

4.6.3 3. How to contribute

1. Choose a topic branch (typically `master`) to start your contribution from
2. Make commits of logical units
3. Ensure that if you contributed code you also include accompanying tests in the `explabox/test` folder
4. Perform all *quality checks* when you are finished
5. Update the *changelog* to state which contributions you made
6. Push your changes to a topic branch and create a *merge request*, describing your contribution
7. Actively watch if your contribution requires any further changes

3.1 Quality checks

When contributing to the `explabox`, you are required to adhere to several quality criteria, as described in the table below. These are checked automatically when making a commit to the `main` branch (using `pre-commit`), and are included in the `Makefile` (run `make quality` and `make coverage` in your terminal, or run `make.bat` if on Windows). In addition, they can be run manually with the command provided in the *Manual check* column below.

Quality	Tool	Description	Manual check
Import order	`isort`	Imports in `.py` files are done in alphabetical order.	`isort --profile=black --line-length=120 --check-only .`
Linter	`black`	Automatic formatting of your `.py` code, weakened to a line length of 120.	`black --line-length=120 --check .`
Linter	`flake8`	Minimal code style quality check, also weakened to a line length of 120.	`flake8 --config .flake8 .`
Security	`bandit`	Software is checked for known security vulnerabilities.	`bandit -r explabox/ --configfile=.bandit.yaml`
Unit and integration testing	`pytest`	Software should be tested in seperate units and in combined pipelines.	`coverage run -m pytest`
`MANIFEST.in` completeness	`check-mar`	Check if all required files are also shipped with the Python package.	`check-manifest`
Documentation linter	`doc8`	Style checks for the documentation files that are used to generate explabox.rtfd.io	`doc8 ./docs`

These tools are automatically included when installing the `explabox` with the `[dev]` and `[all]` options. The documentation linter is installed with the `[doc]`, `[dev]` and `[all]` options.

If you are contributing to the documentation (i.e. editing any `.md` or `.rst` file), ensure you run `make docs` (or `make.bat` for Windows) when you are finished to ensure the updated files are copied to the `/docs` folder.

3.2 Updating `CHANGELOG.md`

Update the changelog using the Keep a Changelog standard, under the `[Unreleased]` section of `CHANGELOG.md`. Contributions should be grouped under `### Added`, `### Changed`, `### Fixed` or `### Removed`. Note that you should not repeat the verb (e.g. *added*) of the group in a bullet point.

3.3 Merge request

Clearly state your contributions when making a merge request. Reference any prior issues you are aiming to solve. If you require *new dependencies or new versions thereof*, also explicitly state why these are required.

Your contribution will be reviewed, potentially requiring changes to the code/documentation you have contributed. When passing all quality checks and the code review, your contribution will become part of the next version of the `explabox`.

3.4 New version release

After a succesful merge request, the developers/maintainers of the `explabox` will ensure your contribution is pushed with the next version of the `explabox`.

4.7 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

explabox, 28
explabox.config, 47
explabox.digestibles, 28
explabox.examine, 32
explabox.explain, 34
explabox.explain.text, 34
explabox.explore, 37
explabox.expose, 38
explabox.expose.text, 38
explabox.expose.text.characters, 41
explabox.expose.text.sentences, 43
explabox.expose.text.words, 43
explabox.ingestibles, 43
explabox.ingestibles.data, 45
explabox.ingestibles.model, 45
explabox.mixins, 47
explabox.ui, 45
explabox.ui.notebook, 45
explabox.utils, 47
explabox.utils.io, 47

INDEX

A

add_typos() (*in module explabox.expose.text.characters*), 41

C

check_requirements() (*explabox.ingestibles.Ingestible method*), 44
check_requirements() (*explabox.mixins.IngestiblesMixin method*), 47
compare_metric() (*explabox.expose.text.Exposer method*), 38
content (*explabox.digestibles.Dataset property*), 29
content (*explabox.digestibles.Descriptives property*), 31
content (*explabox.digestibles.Performance property*), 32
content (*explabox.digestibles.WronglyClassified property*), 32
create_output_dir() (*in module explabox.utils.io*), 47

D

data (*explabox.digestibles.Dataset property*), 29
data (*explabox.ingestibles.Ingestible property*), 44
data (*explabox.mixins.IngestiblesMixin property*), 47
Dataset (*class in explabox.digestibles*), 28
dataset_renderer() (*in module explabox.ui.notebook*), 46
delete_random() (*in module explabox.expose.text.characters*), 41
Descriptives (*class in explabox.digestibles*), 31
descriptives() (*explabox.explore.Explorer method*), 37
descriptives_renderer() (*in module explabox.ui.notebook*), 46

E

Examiner (*class in explabox.examine*), 32
explabox
    module, 28
Explabox (*class in explabox*), 28
explabox.config
    module, 47
explabox.digestibles
    module, 28
explabox.examine
    module, 32
explabox.explain
    module, 34
explabox.explain.text
    module, 34
explabox.explore
    module, 37
explabox.expose
    module, 38
explabox.expose.text
    module, 38
explabox.expose.text.characters
    module, 41
explabox.expose.text.sentences
    module, 43
explabox.expose.text.words
    module, 43
explabox.ingestibles
    module, 43
explabox.ingestibles.data
    module, 45
explabox.ingestibles.model
    module, 45
explabox.mixins
    module, 47
explabox.ui
    module, 45
explabox.ui.notebook
    module, 45
explabox.utils
    module, 47
explabox.utils.io
    module, 47
explain_prediction() (*explabox.explain.text.Explainer method*), 34
Explainer (*class in explabox.explain.text*), 34
Explorer (*class in explabox.explore*), 37
Exposer (*class in explabox.expose.text*), 38

F

filter() (*explabox.digestibles.Dataset method*), 29
format_table() (*in module explabox.ui.notebook*), 46
format_title() (*explabox.ui.notebook.Render method*), 45

G

GBRenderRestyled (*class in explabox.ui.notebook*), 45
get_by_index() (*explabox.digestibles.Dataset method*), 30
get_by_key() (*explabox.digestibles.Dataset method*), 30
get_named_split() (*explabox.ingestibles.Ingestible method*), 44
get_renderer() (*explabox.ui.notebook.Render method*), 46

H

head() (*explabox.digestibles.Dataset method*), 30
html (*explabox.utils.MultipleReturn property*), 47

I

Ingestible (*class in explabox.ingestibles*), 43
IngestiblesMixin (*class in explabox.mixins*), 47
input_space() (*explabox.expose.text.Exposer method*), 39
instances (*explabox.digestibles.Dataset property*), 30
instances() (*explabox.explore.Explorer method*), 37
invariance() (*explabox.expose.text.Exposer method*), 40
is_classifier (*explabox.mixins.ModelMixin property*), 48

K

keys (*explabox.digestibles.Dataset property*), 30

L

labels (*explabox.digestibles.Dataset property*), 30
labels (*explabox.ingestibles.Ingestible property*), 44
labels (*explabox.mixins.IngestiblesMixin property*), 48
labelset (*explabox.ingestibles.Ingestible property*), 44
labelset (*explabox.mixins.IngestiblesMixin property*), 48

M

MAIN_COLOR (*in module explabox.ui.notebook*), 45
mean_score() (*explabox.expose.text.Exposer method*), 40
metrics (*explabox.digestibles.Performance property*), 32
model (*explabox.ingestibles.Ingestible property*), 44
model (*explabox.mixins.IngestiblesMixin property*), 48
ModelMixin (*class in explabox.mixins*), 48
module
    explabox, 28
    explabox.config, 47
    explabox.digestibles, 28
    explabox.examine, 32
    explabox.explain, 34
    explabox.explain.text, 34
    explabox.explore, 37
    explabox.expose, 38
    explabox.expose.text, 38
    explabox.expose.text.characters, 41
    explabox.expose.text.sentences, 43
    explabox.expose.text.words, 43
    explabox.ingestibles, 43
    explabox.ingestibles.data, 45
    explabox.ingestibles.model, 45
    explabox.mixins, 47
    explabox.ui, 45
    explabox.ui.notebook, 45
    explabox.utils, 47
    explabox.utils.io, 47
MultipleReturn (*class in explabox.utils*), 47

P

PACKAGE_LINK (*in module explabox.ui.notebook*), 45
PACKAGE_NAME (*in module explabox.ui.notebook*), 45
Performance (*class in explabox.digestibles*), 31
performance() (*explabox.examine.Examiner method*), 33
prototypes() (*explabox.explain.text.Explainer method*), 35
prototypes_criticisms() (*explabox.explain.text.Explainer method*), 35

R

random_case_swap() (*in module explabox.expose.text.characters*), 41
random_lower() (*in module explabox.expose.text.characters*), 42
random_spaces() (*in module explabox.expose.text.characters*), 42
random_upper() (*in module explabox.expose.text.characters*), 42
raw_html (*explabox.utils.MultipleReturn property*), 47
Render (*class in explabox.ui.notebook*), 45
render_subtitle() (*explabox.ui.notebook.Render method*), 46
repeat_k_times() (*in module explabox.expose.text.sentences*), 43
replace_renderer() (*in module explabox.ui.notebook*), 46
restyle() (*explabox.ui.notebook.RestyleMixin method*), 46
restyle() (*in module explabox.ui.notebook*), 46

RestyleMixin (*class in explabox.ui.notebook*), 46

S

sample() (*explabox.digestibles.Dataset method*), 30
splits (*explabox.ingestibles.Ingestible property*), 44
splits (*explabox.mixins.IngestiblesMixin property*), 48
swap_random() (*in module explabox.expose.text.characters*), 42

T

tail() (*explabox.digestibles.Dataset method*), 31
TERenderRestyled (*class in explabox.ui.notebook*), 46
test (*explabox.ingestibles.Ingestible property*), 44
to_config() (*explabox.utils.MultipleReturn method*), 47
to_lower() (*in module explabox.expose.text.sentences*), 43
to_upper() (*in module explabox.expose.text.sentences*), 43
token_frequency() (*explabox.explain.text.Explainer method*), 36
token_information() (*explabox.explain.text.Explainer method*), 36
train (*explabox.ingestibles.Ingestible property*), 44
TSRenderRestyled (*class in explabox.ui.notebook*), 46

V

validation (*explabox.ingestibles.Ingestible property*), 44

W

wrongly_classified (*explabox.digestibles.WronglyClassified property*), 32
wrongly_classified() (*explabox.examine.Examiner method*), 33
wrongly_classified_renderer() (*in module explabox.ui.notebook*), 46
WronglyClassified (*class in explabox.digestibles*), 32