text_sensitivity

Marcel Robeer

Jan 31, 2023

USING TEXT$_{SENSITIVITY}$

Note: Extension of text_explainability

Uses the **generic architecture** of `text_explainability` to also include tests of **safety** (*how safe it the model in production*, i.e. types of inputs it can handle), **robustness** (*how generalizable the model is in production*, e.g. stability when adding typos, or the effect of adding random unrelated data) and **fairness** (*if equal individuals are treated equally by the model*, e.g. subgroup fairness on sex and nationality).

CHAPTER

ONE

QUICK TOUR

Safety: test if your model is able to handle different data types.

```
from text_sensitivity import RandomAscii, RandomEmojis, combine_generators

# Generate 10 strings with random ASCII characters
RandomAscii().generate_list(n=10)

# Generate 5 strings with random ASCII characters and emojis
combine_generators(RandomAscii(), RandomEmojis()).generate_list(n=5)
```

Robustness: if your model performs equally for different entities ...

```
from text_sensitivity import RandomAddress, RandomEmail

# Random address of your current locale (default = 'nl')
RandomAddress(sep=', ').generate_list(n=5)

# Random e-mail addresses in Spanish ('es') and Portuguese ('pt'), and include from which␣
↪country the e-mail is
RandomEmail(languages=['es', 'pt']).generate_list(n=10, attributes=True)
```

... and if it is robust under simple perturbations.

```
from text_sensitivity import compare_accuracy
from text_sensitivity.perturbation import to_upper, add_typos

# Is model accuracy equal when we change all sentences to uppercase?
compare_accuracy(env, model, to_upper)

# Is model accuracy equal when we add typos in words?
compare_accuracy(env, model, add_typos)
```

Fairness: see if performance is equal among subgroups.

```
from text_sensitivity import RandomName

# Generate random Dutch ('nl') and Russian ('ru') names, both 'male' and 'female' (+ return␣
↪attributes)
RandomName(languages=['nl', 'ru'], sex=['male', 'female']).generate_list(n=10,␣
↪attributes=True)
```

CHAPTER

TWO

USING TEXT_SENSITIVITY

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

Example Usage
: An extended usage example.

text_sensitivity API reference
: A reference to all classes and functions included in the `text_sensitivity`.

CHAPTER

THREE

DEVELOPMENT

text_sensitivity @ 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
: Contributors to the open-source project and contribution guidelines.

CHAPTER

FOUR

CITATION

```bibtex
@misc{text_sensitivity,
  title = {Python package text\_sensitivity},
  author = {Marcel Robeer},
  howpublished = {\url{https://git.science.uu.nl/m.j.robeer/text_sensitivity}},
  year = {2021}
}
```

CHAPTER

FIVE

CREDITS

- Edward Ma. NLP Augmentation. 2019.
- Daniele Faraglia and other contributors. Faker. 2012.
- Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin and Sameer Singh. Beyond Accuracy: Behavioral Testing of NLP models with CheckList. *Association for Computational Linguistics* (ACL). 2020.

5.1 Installation

Installation of `text_sensitivity` requires `Python 3.8` or higher.

5.1.1 1. Python installation

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

5.1.2 2. Installing `text_sensitivity`

`text_sensitivity` can be installed:

- *using pip*: *pip3 install* (released on [PyPI](https://pypi.org/project/text-sensitivity/))
- *locally*: cloning the repository and using `python3 setup.py install`

Using `pip`

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

```
user@terminal:~$ pip3 install text_sensitivity
Collecting text_sensitivity
...
Installing collected packages: text-sensitivity
Successfully installed text-sensitivity
```

Locally

1. Download the folder from `GitLab/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 `text_sensitivity` in.
3. In the main folder (containing the `setup.py` file) run:
    - `python3 setup.py install`, or
    - `python setup.py install`.

```
user@terminal:~$ cd ~/text_sensitivity
user@terminal:~/text_explanability$ python3 setup.py install
running install
running bdist_egg
running egg_info
...
Finished processing dependencies for text-sensitivity
```

5.2 Example Usage

5.2.1 Dependencies

Like `text_explainability`, `text_sensitivity` uses instances and machine learning models wrapped with the InstanceLib library.

5.2.2 Dataset and model

We manually create a `TextEnvironment`, that holds both our ground-truth labels (`.labels`) and our instances (`.dataset`). Next, we fit a simple `sklearn` model that predicts whether the instances (sentence-length strings) contain punctuation or not.

```
# Create a simple dataset (classify whether strings contain punctuation or not)
from instancelib.environment.text import TextEnvironment

instances = ['This is his example instance, not HERS!',
             'An example sentence for you?!',
             'She has her own sentence.',
             'Provide him with something without any punctuation',
             'RANDOM UPPERCASESTRING3']
labels = ['punctuation', 'punctuation', 'punctuation', 'no_punctuation', 'no_punctuation
↪']

env = TextEnvironment.from_data(indices=list(range(len(instances))),
                                data=instances,
                                target_labels=list(set(labels)),
                                ground_truth=[[label] for label in labels],
```

(continues on next page)

(continued from previous page)

```
                                vectors=[])

# Create sklearn model with pipeline
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

p = Pipeline([('vect', CountVectorizer()),
              ('rf', MultinomialNB())])

# Wrap sklearn model
from text_explainability import import_model
import_model(p, env)
```

5.2.3 Using Text Sensitivity

Text Sensitivity is used for *robustness testing* (verifying if a model can handle all types of string data and whether its predictions are invariant to minor changes) and *fairness testing* (comparing model performance on subgroups).

Robustness

A robust text model should be able to handle different types of input strings (e.g. ASCII, emojis) and be invariant to minor changes in inputs (e.g. converting a string to uppercase, adding an unrelated string or users making typos).

Generating random data

Random strings can be used for testing if a model is able to handle ass sorts of inputs:

```
from text_sensitivity import (RandomData, RandomDigits, RandomAscii, RandomEmojis,
                              RandomWhitespace, RandomCyrillic, combine_generators)

# Generate 10 instances with all printable characters
RandomData().generate_list(n=10, min_length=5, max_length=50)

# Generate 5 instances containing only digits
RandomDigits(seed=1).generate_list(n=5)

# Generate 15 instances, combining emojis, whitespace characters and ASCII characters
random_generator = combine_generators(RandomAscii(), RandomEmojis(), RandomWhitespace())
random_generator.generate_list(n=15)

# Generate 20 instances with random ASCII characters, whitespace and Russian (Cyrillic)␣
↪characters
ascii_cyrillic_generator = combine_generators(RandomAscii(), RandomWhitespace(),␣
↪RandomCyrillic(languages='ru'))
ascii_cyrillic_generator.generate_list(n=20)
```

Invariance testing

A very simple method for invariance testing, is assessing whether the model performs the same on a metric (e.g. accuracy, precision or recall) before and after applying a perturbation. For example, let us compare whether the model retains the same performance when converting all instances to lowercase:

```
from text_sensitivity.test import compare_accuracy
from text_sensitivity.perturbation.sentences import to_lower

compare_accuracy(env, model, to_lower)
```

Similarly, we can check whether precision scores are the same if we add an unrelated string after each sentence:

```
from text_sensitivity.test import compare_precision
from text_sensitivity.perturbation.base import OneToOnePerturbation

perturbation_fn = OneToOnePerturbation.from_string(suffix='This should not affect scores
↪')
compare_precision(env, model, perturbation_fn)
```

Under the hood, `text_sensitivity.test` uses `text_sensitivity.perturbation` to perturb instances (`instancelib.instances.text.TextInstance` or `str`), and generates the new instances and labels for the original instance (e.g. 'not_upper') and the new instance(s) (e.g. 'upper').

```
from text_sensitivity.perturbation.sentences import to_upper, repeat_k_times
from text_sensitivity.perturbation.characters import random_case_swap, random_spaces,␣
↪swap_random, add_typos

sample = 'This is his example string, made especially for HER!'

# Convert the sample string to all upper
list(to_upper()(sample))

# Repeat the string 'test' n times
list(repeat_k_times(n=3)('test'))
list(repeat_k_times(n=7, connector='\n')('test'))

# Randomly swap the character case (lower to upper or vice versa) in sample
list(random_case_swap()(sample))

# Add random spaces to words within a sentence, or swap characters randomly within a␣
↪word (excluding stopwords and uppercase words) to sample
list(random_spaces(n=5)(sample))
list(swap_random(n=10, stopwords=['the' , 'is', 'of'], include_upper_case=False)(sample))

# Add typos (based on QWERTY keyboard) to sample
list(add_typos(n=10, stopwords=['the' , 'is', 'of'], include_numeric=False, include_
↪special_char=False)(sample))
```

Fairness

TODO: Write up fairness.

Generating random data

Data for entities can be generated in the same manner as random strings:

```
from text_sensitivity import (RandomCity, RandomCountry, RandomName)

# Generates data for the current locale, e.g. if it is 'nl' it generates country names in␣
↪Dutch and cities in the Netherlands
RandomCity().generate_list(n=10)

# If you specify the locale, it can generate the entity (e.g. country) for multiple␣
↪languages
RandomCountry(languages=['nl', 'de', 'fr', 'jp']).generate_list(n=15)
```

Unlike random strings, random entities can also output the corresponding attribute labels for the generated data

```
# For example, generated Dutch and Russian male and female names, and output which␣
↪language and sex they are
generator = RandomName(languages=['nl', 'ru'], sex=['male', 'female'], seed=5)
generator.generate_list(n=10, attributes=True)

# The same data can also be captured in an instancelib.InstanceProvider and instancelib.
↪LabelProviders
generator.generate(n=10, attributes=True)
```

Other random entities that can be generated are dates, street addresses, emails, phone numbers, price tags and crypto names:

```
# Dates
from text_sensitivity import RandomYear, RandomMonth, RandomDay, RandomDayOfWeek

print(RandomYear().generate_list(n=3))
print(RandomMonth(languages=['nl', 'en']).upper().generate_list(n=6))  # use .upper() to␣
↪generate all uppercase or .lower() for all lower
print(RandomDay().generate_list(n=3))
print(RandomDayOfWeek().sentence().generate_list(n=3))  # use .sentence() for all␣
↪sentencecase or .title() for titlecase

# Street addresses, emails, phone numbers, price tags and crypto names
from text_sensitivity import RandomAddress, RandomEmail, RandomPhoneNumber,␣
↪RandomPriceTag, RandomCryptoCurrency

print(RandomAddress(sep=', ').generate_list(n=5))
print(RandomEmail(languages=['es', 'pt']).generate_list(n=10, attributes=True))
print(RandomPhoneNumber().generate_list(n=5))
print(RandomPriceTag(languages=['ru', 'de', 'it', 'br']).generate_list(n=10))
print(RandomCryptoCurrency().generate_list(n=3))
```

Generating data from patterns

These entities, or your own lists, can be used to generate strings for locally testing model robustness/fairness. Text within curly braces ({}) is replaced, and attribute are added to each perturbed instance. The text outside of the curly braces remains the same. Examples of patterns that can be put between curly braces are:

- `{a|b|c}` generates a list with elements `a`, `b` and `c`.
- `{city}` uses `RandomCity()` (in current locale) to generate `n` random cities. For a full list of default patterns see `from text_sensitivity import default_patterns; default_patterns()`.
- `{custom_entity_name}` with keyword argument `custom_entity_name=['this', 'is', 'cool]` generates a list with elements `this`, `is`, `cool`.

```
from text_sensitivity import from_pattern

# Generate a list ['This is his house', 'This was his house', 'This is his car', 'This was␣
↪his car', ...]:
from_pattern('This {is|was} his {house|car|boat}')

# Generate a list ['His home town is Eindhoven.', 'Her home town is Eindhoven.',  'His home␣
↪town is Meerssen.', ...]. By default uses `RandomCity()` to generate the city name.
from_pattern('{His|Her} home town is {city}.')

# Override the 'city' default with your own list ['Amsterdam', 'Rotterdam', 'Utrecht']:
from_pattern('{His|Her} home town is {city}.', city=['Amsterdam', 'Rotterdam', 'Utrecht
↪'])
```

In addition, modifiers can be added before a semicolon (`:`) within a curly brace to modify the generated data. Example modifiers are:

- `{lower:address}` generates addresses (`RandomAddress()` for current locale) in all-lowercase
- `{upper:name}` generates full name (`RandomName()` for current locale) in all-uppercase
- `{sentence:day_of_week}` generates day of week (`RandomDayOfWeek()` for current locale) in sentencecase.
- `{title:country}` generates country names (`RandomCountry()` in locale language) in titlecase.

```
# Apply lower case to the first argument and uppercase to the last, getting ['Vandaag,␣
↪donderdag heeft Sanne COLIN gebeld op +31612351983!', ..., 'Vandaag, maandag heeft Nora␣
↪SEPP gebeld op +31612351983!', ...]
from_pattern('Vandaag, {lower:day_of_week}, heeft {first_name} {upper:first_name} gebeld␣
↪op {phone_number}!', n=5)
```

5.3 text_sensitivity

Subpackages:

5.3.1 text_sensitivity.data

All randomly generated data, and data for lookups (e.g. word lists).

Subpackages:

text_sensitivity.data.lists

text_sensitivity.data.random

Generate random data for robustness and sensitivity testing.

Submodules:

text_sensitivity.data.random.entity module

Generation of random entities (e.g. names, telephone numbers) for given languages.

class text_sensitivity.data.random.entity.**CityByPopulationMixin**

Bases: Readable

add_likelihood_to_cities()

Add likelihood to cities, based on population.

static cities_by_population(*cities*, *country_code*)

Add population scores to each city in a country.

Parameters

- **cities** (*List[str]*) – Current list of cities. If no replacement is found, this will be returned back.
- **country_code** (*str*) – Two-letter country code (e.g. 'nl').

class text_sensitivity.data.random.entity.**RandomAddress**(*languages=<Proxy at 0x7f9c80e8e440 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>*, *likelihood_based_on_city_population=True*, *sep='\\n'*, *seed=0*)

Bases: *RandomEntity*, *CityByPopulationMixin*

Generate random cities in (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **likelihood_based_on_city_population** (*bool*) –
- **sep** (*str*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomCity**(*languages=<Proxy at 0x7f9c80e8e6c0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>*, *likelihood_based_on_city_population=True*, *seed=0*)

Bases: *RandomEntity*, *CityByPopulationMixin*

Generate random cities in (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **likelihood_based_on_city_population** (*bool*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomCountry**(*languages=<Proxy at 0x7f9c80e8e900 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>*, *seed=0*)

Bases: *RandomEntity*

Generate random countries for (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomCryptoCurrency**(*seed=0*)

Bases: *RandomEntity*

Generate random cryptocurrency names.

Parameters

seed (*int*) –

class text_sensitivity.data.random.entity.**RandomCurrencySymbol**(*seed=0*)

Bases: *RandomEntity*

Generate random currency symbols.

Parameters

seed (*int*) –

class text_sensitivity.data.random.entity.**RandomDay**(*seed=0*)

Bases: *RandomEntity*

Generate random day of the month.

Parameters

seed (*int*) –

class text_sensitivity.data.random.entity.**RandomDayOfWeek**(*languages=<Proxy at 0x7f9c80e918c0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>*, *seed=0*)

Bases: *RandomEntity*

Generate random day of week in (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomEmail**(*languages=<Proxy at 0x7f9c80e911c0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, seed=0*)

Bases: *RandomEntity*

Generate random e-mail addresses for (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomEntity**(*languages=<Proxy at 0x7f9c813326c0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, providers=['person'], fn_name='name', attribute='fn', attribute_rename=None, sep=\\n', seed=0*)

Bases: Readable, SeedMixin, CaseMixin

Base class to generate entity data for (a) given language(s).

Example

Generate a 10 random English names entity using package *faker*:

```
>>> RandomEntity(locale='en', providers=['person'], fn_name='name').generate_
↪list(n=10)
```

Parameters

- **languages** (*Union[str, List[str]], optional*) – Languages to generate data from. Defaults to your current locale (see *get_locale()*).
- **providers** (*List[str], optional*) – Providers from *faker* used in generation. Defaults to ['person'].
- **fn_name** (*Union[str, List[str]], optional*) – Function name(s) to call for each generator. Defaults to 'name'.
- **attribute** (*str, optional*) – Name of additional attribute (other than language). Defaults to 'fn'.
- **attribute_rename** (*Optional[Callable[[str], str]], optional*) – Rename function for attribute value. Defaults to None.
- **sep** (*str, optional*) – Separator to replace 'n' character with. Defaults to 'n'.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

generate(*n*, *attributes=False*)

Generate n instances of random data.

Parameters

- **n** (*int*) – Number of instances to generate.

- **attributes** (*bool, optional*) – Include attributes (language, which function was used, etc.) or not. Defaults to False.

Returns

Provider containing generated instances (if attributes = False). Tuple[TextInstanceProvider, Dict[str, MemoryLabelProvider]]: Provider and corresponding attribute

labels (if attributes = True).

Return type
TextInstanceProvider

generate_list(*n*, *attributes=False*)

Generate n instances of random data and return as list.

Parameters

- **n** (*int*) – Number of instances to generate.
- **attributes** (*bool, optional*) – Include attributes (language, which function was used, etc.) or not. Defaults to False.

Returns
Generated instances (if attributes = False). Tuple[List[str], Dict[str, str]]: Generated instances and corresponding attributes (if attributes = True).

Return type
List[str]

class text_sensitivity.data.random.entity.**RandomFirstName**(*languages=<Proxy at 0x7f9c80e8ed40 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, sex=['male', 'female'], seed=0*)

Bases: *RandomEntity*

Generate random first names for (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **sex** (*List[str]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomLastName**(*languages=<Proxy at 0x7f9c80e8ef80 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, seed=0*)

Bases: *RandomEntity*

Generate random last names for (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomLicensePlate**(*seed=0*)

Bases: *RandomEntity*

Generate random license plates for a given country.

Parameters
seed (*int*) –

class text_sensitivity.data.random.entity.**RandomMonth**(*languages=<Proxy at 0x7f9c80e91640 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, seed=0*)

Bases: *RandomEntity*

Generate random month name in (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomName**(*languages=<Proxy at 0x7f9c80e8eb00 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, sex=['male', 'female'], seed=0*)

Bases: *RandomEntity*

Generate random full names for (a) given language(s).

Parameters

- **languages** (*Union[str, List[str]]*) –
- **sex** (*List[str]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomPhoneNumber**(*languages=<Proxy at 0x7f9c80e913c0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, seed=0*)

Bases: *RandomEntity*

Generate random phone numbers for (a) given language(s) / country.

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomPriceTag**(*languages=<Proxy at 0x7f9c80e91ac0 wrapping 'nl' at 0x7f9ca92315f0 with factory <function lazy.<locals>.<lambda>>>, seed=0*)

Bases: *RandomEntity*

Generate random pricetag names in (a) given languages' currency.

Parameters

- **languages** (*Union[str, List[str]]*) –
- **seed** (*int*) –

class text_sensitivity.data.random.entity.**RandomYear**(*seed=0*)

Bases: *RandomEntity*

Generate random year.

Parameters
seed (*int*) –

text_sensitivity.data.random.string module

Generate random strings from characters/strings.

class text_sensitivity.data.random.string.**RandomAscii**(*seed=0*)

Bases: *RandomString*

Generate random ASCII characters.

Parameters
seed (*int*) –

class text_sensitivity.data.random.string.**RandomCyrillic**(*languages='ru'*, *upper=True*, *lower=True*, *seed=0*)

Bases: *RandomString*

Generate containing random Cyrillic characters.

Can generate text in Bulgarian ('bg'), Macedonian ('mk'), Russian ('ru'), Serbian ('sr'), Ukrainian ('uk'), and all combinations thereof.

Parameters

- **languages** (*Union[List[str], str], optional*) – Cyrillic languages to select. Defaults to 'ru'.
- **upper** (*bool, optional*) – Whether to include
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- **lower** (*bool*) –

Raises

- **ValueError** – Either upper or lower should be True.
- **ValueError** – One of the selected languages is unknown.

class text_sensitivity.data.random.string.**RandomDigits**(*seed=0*)

Bases: *RandomString*

Generate strings containing random digits.

Parameters
seed (*int*) –

class text_sensitivity.data.random.string.**RandomEmojis**(*seed=0*, *base=True*, *dingbats=True*, *flags=True*, *components=True*)

Bases: *RandomString*

Generate strings containing a subset of random unicode emojis.

Parameters

- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- **base** (*bool, optional*) – Include base emojis (e.g. smiley face). Defaults to True.
- **dingbats** (*bool, optional*) – Include dingbat emojis. Defaults to True.
- **flags** (*bool, optional*) – Include flag emojis. Defaults to True.
- **components** (*bool, optional*) – Include emoji components (e.g. skin color modifier or country flags). Defaults to True.

Raises

ValueError – At least one of *base*, *dingbats*, *flags* should be True.

class text_sensitivity.data.random.string.**RandomLower**(*seed=0*)

Bases: *RandomString*

Generate random ASCII lowercase characters.

Parameters

seed (*int*) –

class text_sensitivity.data.random.string.**RandomPunctuation**(*seed=0*)

Bases: *RandomString*

Generate strings containing random punctuation characters.

Parameters

seed (*int*) –

class text_sensitivity.data.random.string.**RandomSpaces**(*seed=0*)

Bases: *RandomString*

Generate strings with a random number of spaces.

Parameters

seed (*int*) –

class text_sensitivity.data.random.string.**RandomString**(*seed=0*, *options='0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHI -./:;<=>?@[\\\\]^_`{|}~ \\t\\n\\r\\x0b\\x0c'*)

Bases: Readable, SeedMixin

Base class for random data (string) generation.

Parameters

- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- **options** (*Union[str, List[str]], optional*) – Characters or strings to generate data from. Defaults to string.printable.

generate(*n*, *min_length=0*, *max_length=100*)

Generate n instances of random strings.

Example

Create a TextInstanceProvider containing n=10 strings of random characters from *'12345xXyY!?'* between length 3 and 10:

```
>>> RandomString(seed=0, options='12345xXyY!?').generate_list(n=10, min_
↪length=3, max_length=10)
```

Parameters

- **n** (*int*) – Number of instances to generate.
- **min_length** (*int, optional*) – Minimum length of random instance. Defaults to 0.
- **max_length** (*int, optional*) – Maximum length of random instance. Defaults to 100.

Raises

ValueError – *min_length* should be smaller than *max_length*.

Returns

Provider containing generated instances.

Return type

TextInstanceProvider

generate_list(*n*, *min_length=0*, *max_length=100*)

Generate n instances of random strings and return as list.

Example

Generate a list of random characters from *u'ABCabcU0001F600'* between length 10 and 50 (n=10 strings):

```
>>> RandomString(seed=0, options=u'ABCabc\U0001F600').generate_list(n=10, min_
↪length=10, max_length=50)
```

Parameters

- **n** (*int*) – Number of instances to generate.
- **min_length** (*int, optional*) – Minimum length of random instance. Defaults to 0.
- **max_length** (*int, optional*) – Maximum length of random instance. Defaults to 100.

Raises

ValueError – *min_length* should be smaller than *max_length*.

Returns

List containing generated instances.

Return type

List[str]

class text_sensitivity.data.random.string.**RandomUpper**(*seed=0*)

Bases: *RandomString*

Generate random ASCII uppercase characters.

Parameters

seed (*int*) –

class text_sensitivity.data.random.string.**RandomWhitespace**(*seed=0*)

Bases: *RandomString*

Generate strings with a random number whitespace characters.

Parameters
seed (*int*) –

text_sensitivity.data.random.string.**combine_generators**(**generators*, *seed=None*)

Combine muliple random string generators into one.

Parameters

- ***generators** – Generators to combine.
- **seed** (*Optional[int]*) – Seed value for new generator. If None picks a random seed from the generators. Defaults to None.

Return type
RandomString

Example

Make a generator that generates random punctuation, emojis and ASCII characters:

```
>>> new_generator = combine_generators(RandomPunctuation(), RandomEmojis(),␣
↪RandomAscii())
```

Returns
Generator with all generator options combined.

Return type
RandomString

Parameters
seed (*Optional[int]*) –

Submodules:

text_sensitivity.data.generate module

Generate data from a pattern, e.g. *'{He|She} lives in {city}.'*

text_sensitivity.data.generate.**default_patterns**()

Overview of all default patterns.

Return type
List[str]

text_sensitivity.data.generate.**from_pattern**(*pattern*, *n=3*, *seed=0*, ***kwargs*)

Generate data from a pattern.

Examples

Generate a list [‘This is his house’, ‘This was his house’, ‘This is his car’, ‘This was his car’, . . .]:

```
>>> from_pattern('This {is|was} his {house|car|boat}')
```

Generate a list [‘His home town is Eindhoven.’, ‘Her home town is Eindhoven.’, ‘His home town is Meerssen.’, . . .]. By default uses *RandomCity()* to generate the city name.

```
>>> from_pattern('{His|Her} home town is {city}.')
```

Override the ‘city’ default with your own list [‘Amsterdam’, ‘Rotterdam’, ‘Utrecht’]:

```
>>> from_pattern('{His|Her} home town is {city}.', city=['Amsterdam', 'Rotterdam',
↪'Utrecht'])
```

Apply lower case to the first argument and uppercase to the last, getting [‘Vandaag, donderdag heeft Sanne COLIN gebeld!’, . . . , ‘Vandaag, maandag heeft Nora SEPP gebeld!’, . . .] for five random elements of each:

```
>>> from_pattern('Vandaag, {lower:day_of_week}, heeft {first_name} {upper:first_
↪name} gebeld!', n=5)
```

Parameters

- **pattern** (*str*) – String containing pattern.
- **n** (*int, optional*) – Number of elements to generate for each element, when generator is random. Defaults to 3.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

Returns

Generated instances and corresponding labels.

Return type

Tuple[TextInstanceProvider, Dict[LT, MemoryLabelProvider]]

text_sensitivity.data.generate.**options_from_brackets**(*string*, *n=3*, *seed=0*, ***kwargs*)

Generate options from string.

Example

Generate random list of houses:

```
>>> options_from_brackets('I have {number} houses!', number=[5, 10, 100, 5000])
```

Parameters

- **string** (*str*) – String with curly braces.
- **n** (*int, optional*) – Number of elements to generate for each option. Defaults to 3.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

Returns

Strings with elements generated.

Return type
List[str]

text_sensitivity.data.wordlist module

Select data from a list of words, optionally with a probability to choose each element.

class text_sensitivity.data.wordlist.**WordList**(*wordlist*, *main_column=None*, *attribute_column=None*, *seed=0*)

Bases: Readable, SeedMixin, CaseMixin

Capture data in wordlist.

Parameters

- **wordlist** (*pd.DataFrame*) – Dataframe containing a column with data (e.g. city names).
- **main_column** (*Optional[Label], optional*) – Column containing data. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Column containing attributes. If None defaults to the main column.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

filter(*column*, *values*)

Filter the wordlist column if it is in values.

Parameters

- **column** (*Label*) – Column to filter.
- **values** (*Union[Label, List[Label]]*) – Values to filter.

Returns
Self.

Return type
WordList

classmethod from_csv(*filename*, *main_column=None*, *attribute_column=None*, *seed=0*, **args*, ***kwargs*)

Create a *WordList* from a CSV file.

Parameters

- **filename** (*str*) – Filename.
- **main_column** (*Optional[Label], optional*) – Data column. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Attribute column. Defaults to None.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- ****kwargs** – Optional arguments passed to *pandas.read_csv()*.

Returns
WordList class.

Return type
WordList

classmethod from_dict(**args*, ***kwargs*)

Alias for *WordList.from_dictionary()*.

classmethod from_dictionary(*wordlist*, *key_name='key'*, *value_name='value'*, *value_as_main=False*, *seed=0*)

Create a *WordList* from a dictionary.

Example

Create list of pronouns with genders:

```
>>> wl = WordList.from_dictionary({'he': 'male', 'she': 'female', 'they':
↪'neuter'},
...                               key_name='pronoun',
...                               value_name='gender')
```

Parameters

- **wordlist** (*Dict*) – Dictionary of elements and corresponding attribute.
- **key_name** (*Label, optional*) – Name of keys. Defaults to 'key'.
- **value_name** (*Label, optional*) – Name of values. Defaults to 'value'.
- **value_as_main** (*bool, optional*) – Whether data is in the key column (False) or value column (True). Defaults to False.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

Returns
WordList class.

Return type
WordList

classmethod from_excel(*filename*, *main_column=None*, *attribute_column=None*, *seed=0*, **args*, ***kwargs*)

Create a *WordList* from an Excel (*.xls* or *.xlsx*) file.

Parameters

- **filename** (*str*) – Filename.
- **main_column** (*Optional[Label], optional*) – Data column. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Attribute column. Defaults to None.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- ****kwargs** – Optional arguments passed to *pandas.read_excel()*.

Returns
WordList class.

Return type
WordList

classmethod from_file(*filename*, *main_column=None*, *attribute_column=None*, *seed=0*, **args*, ***kwargs*)

Create a *WordList* from a file.

The file type is inferred based on the file extension.

Parameters

- **filename** (*str*) – Filename.
- **main_column** (*Optional[Label], optional*) – Data column. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Attribute column. Defaults to None.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- ****kwargs** – Optional arguments passed to *pandas* reader.

Returns
WordList class.

Return type
WordList

classmethod from_json(*filename*, *main_column=None*, *attribute_column=None*, *seed=0*, **args*, ***kwargs*)

Create a *WordList* from a JSON file.

Parameters

- **filename** (*str*) – Filename.
- **main_column** (*Optional[Label], optional*) – Data column. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Attribute column. Defaults to None.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- ****kwargs** – Optional arguments passed to *pandas.read_json()*.

Returns
WordList class.

Return type
WordList

classmethod from_list(*wordlist*, *name='words'*, *seed=0*)

Create a *WordList* from a list of strings.

Example

Create list of city names and pick one random element:

```
>>> wl = WordList.from_list(['Amsterdam', 'Rotterdam', 'Utrecht'], name='city')
>>> wl.generate_list(n=1)
```

Parameters

- **wordlist** (*List[str]*) – List of strings.
- **name** (*Label, optional*) – Name of attribute. Defaults to 'words'.

- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.

Returns
WordList class.

Return type
WordList

classmethod from_pickle(*filename*, *main_column=None*, *attribute_column=None*, *seed=0*, **args*, ***kwargs*)

Create a *WordList* from a Pickled (*.pkl*) file.

Parameters

- **filename** (*str*) – Filename.
- **main_column** (*Optional[Label], optional*) – Data column. Defaults to None.
- **attribute_column** (*Optional[Label], optional*) – Attribute column. Defaults to None.
- **seed** (*int, optional*) – Seed for reproducibility. Defaults to 0.
- ****kwargs** – Optional arguments passed to *pandas.read_pickle()*.

Returns
WordList class.

Return type
WordList

generate_list(*n=None*, *attributes=False*, *likelihood_column=None*)

Generate a random list of *n* elements.

Parameters

- **n** (*Optional[int], optional*) – Number of elements to generate. Defaults to None.
- **attributes** (*bool, optional*) – Include attributes or not. Defaults to False.
- **likelihood_column** (*Optional[Label], optional*) – Attribute to determine likelihood on. Defaults to None.

Returns
Wordlist elements (up to *n*).

Return type
List[str]

get(*sort_by=None*, *attributes=False*, ***sort_kwargs*)

Get all elements in wordlist.

Parameters

- **sort_by** (*Optional[Label], optional*) – Label to sort on (e.g. frequency). Defaults to None.
- **attributes** (*bool, optional*) – Include attributes or not. Defaults to False.

Returns
Wordlist elements.

Return type
List[str]

reset()

Reset wordlist.

class text_sensitivity.data.wordlist.**WordListGetterMixin**

Bases: object

filter(**args*, ***kwargs*)

Wrapper of *WordList.filter()*.

generate_list(**args*, ***kwargs*)

Wrapper of *WordList.generate_list()*.

get(**args*, ***kwargs*)

Get item in wordlist.

reset()

Wrapper of *WordList.reset()*.

5.3.2 text_sensitivity.perturbation

Apply perturbation to one or multiple (tokenized) strings.

Submodules:

text_sensitivity.perturbation.base module

Apply perturbations to TextInstances and/or strings, generating one or many new instances.

class text_sensitivity.perturbation.base.**OneToManyPerturbation**(*perturbation_function*)

Bases: *Perturbation*

Apply a perturbation function to a single *TextInstance*, getting a multiple results per instance.

Parameters

perturbation_function (*Callable*) – Perturbation function to apply, including attribute label of original instance and the resulting instances. Should return None if no perturbation has been applied.

classmethod from_dictionary(*dictionary*, *label_from*, *label_to*, *n=10*, *tokenizer=<function word_tokenizer>*, *detokenizer=<function word_detokenizer>*)

Construct a *OneToManyPerturbation* from a dictionary.

Example

Replace the word 'good' (positive) with 'bad', 'mediocre', 'terrible' (negative) up to 5 times in each instance. The default tokenizer/detokenizer assumes word-level tokens:

```
>>> replacements = {'good': ['bad', 'mediocre', 'terrible']}
>>> OneToManyPerturbation.from_dictionary(replacement,
>>>                                       n=5,
>>>                                       label_from='positive',
>>>                                       label_to='negative')
```

Parameters

- **dictionary** (*Dict[str, List[str]]*) – Lookup dictionary to map tokens (e.g. words, characters).
- **label_from** (*LT*) – Attribute label of original instance (left-hand side of dictionary).
- **label_to** (*LT*) – Attribute label of perturbed instance (right-hand side of dictionary).
- **n** (*int, optional*) – Number of instances to generate. Defaults to 10.
- **tokenizer** (*Callable, optional*) – Function to tokenize instance data (e.g. words, characters). Defaults to default_tokenizer.
- **detokenizer** (*Callable, optional*) – Function to detokenize tokens into instance data. Defaults to default_detokenizer.

classmethod from_function(*function*, *label_from='original'*, *label_to='perturbed'*, *n=10*, *perform_once=False*)

Construct a *OneToManyPerturbation* from a perturbation applied to a string.

Parameters

- **function** (*Callable[[str], Optional[Union[str, Sequence[str]]]]*) – Function to apply to each string. Return None if no change was applied.
- **label_from** (*LT, optional*) – Attribute label of original instance. Defaults to 'original'.
- **label_to** (*LT, optional*) – Attribute label of perturbed instance. Defaults to 'perturbed'.
- **n** (*int, optional*) – Number of instances to generate. Defaults to 10.
- **perform_once** (*bool, optional*) – If the n parameter is in class construction perform once. Defaults to False.

classmethod from_nlpaug(*augmenter*, *label_from='original'*, *label_to='perturbed'*, *n=10*, ***augment_kwargs*)

Construct a *OneToManyPerturbation* from a nlpaug Augmenter.

Example

Add *n=5* versions of keyboard typing mistakes to lowercase characters in a sentence using *nlpaug.augmenter.char.KeyboardAug()*:

```
>>> import nlpaug.augmenter.char as nac
>>> augmenter = nac.KeyboardAug(include_upper_case=False,
>>>                             include_special_char=False)
>>> OneToManyPerturbation.from_nlpaug(augmenter, n=5, label_from='no_typos',␣
↪label_to='typos')
```

Parameters

- **augmenter** (*Augmenter*) – Class with *.augment()* function applying a perturbation to a string.
- **label_from** (*LT, optional*) – Attribute label of original instance. Defaults to 'original'.
- **label_to** (*LT, optional*) – Attribute label of perturbed instance. Defaults to 'perturbed'.
- **n** (*int, optional*) – Number of instances to generate. Defaults to 10.

- ****augment_kwargs** – Optional arguments passed to *.augment()* function.

perturb(*instance*)

Apply a perturbation function to a single *TextInstance*, getting a multiple results per instance.

Parameters

- **perturbation_function** (*Callable*) – Perturbation function to apply, including attribute label of original instance and the resulting instances. Should return None if no perturbation has been applied.
- **instance** (*TextInstance*) –

Returns

None if no perturbation has been applied.
Otherwise a sequence of perturbed TextInstances, and attribute labels for the original and perturbed instances.

Return type
Optional[Sequence[Tuple[TextInstance, Sequence[Tuple[KT, LT]]]]]

class text_sensitivity.perturbation.base.**OneToOnePerturbation**(*perturbation_function*)

Bases: *Perturbation*

Apply a perturbation function to a single *TextInstance*, getting a single result per instance.

Parameters
perturbation_function (*Callable*) – Perturbation function to apply, including attribute label of original instance and the resulting instance. Should return None if no perturbation has been applied.

classmethod from_dictionary(*dictionary*, *label_from*, *label_to*, *tokenizer=<function word_tokenizer>*, *detokenizer=<function word_detokenizer>*)

Construct a *OneToOnePerturbation* from a dictionary.

Example

Replace the word 'a' or 'an' (indefinite article) with 'the' (definite article) in each instance. The default tokenizer/detokenizer assumes word-level tokens:

```
>>> replacements = {'a': 'the',
>>>                 'an': 'the'}
>>> OneToOnePerturbation.from_dictionary(replacement,
>>>                                       label_from='indefinite',
>>>                                       label_to='definite')
```

Replace the character '.' with '!' (character-level replacement): >>> from text_explainability import character_tokenizer, character_detokenizer >>> OneToOnePerturbation.from_dictionary({'.': '!'}, >>> label_from='not_excited', >>> label_to='excited', >>> tokenizer=character_tokenizer, >>> detokenizer=character_detokenizer)

Parameters

- **dictionary** (*Dict[str, str]*) – Lookup dictionary to map tokens (e.g. words, characters).
- **label_from** (*LT*) – Attribute label of original instance (left-hand side of dictionary).
- **label_to** (*LT*) – Attribute label of perturbed instance (right-hand side of dictionary).

- **tokenizer** (*Callable, optional*) – Function to tokenize instance data (e.g. words, characters). Defaults to default_tokenizer.
- **detokenizer** (*Callable, optional*) – Function to detokenize tokens into instance data. Defaults to default_detokenizer.

classmethod from_list(*mapping_list*, *label_from='original'*, *label_to='perturbed'*, *tokenizer=<function word_tokenizer>*, *detokenizer=<function word_detokenizer>*)

Construct a *OneToOnePerturbation* from a list.

A function is constructed that aims to map any value in the list to any other value in the list.

Example

For example, if list *['Amsterdam', 'Rotterdam', 'Utrecht']* is provided it aims to map 'Amsterdam' to 'Rotterdam' or 'Utrecht', 'Rotterdam' to 'Amsterdam' to 'Utrecht' and 'Utrecht' to 'Rotterdam' or 'Amsterdam'. If None of these is possible, it returns None.

```
>>> map_list = ['Amsterdam', 'Rotterdam', 'Utrecht']
>>> OneToOnePerturbation.from_list(map_list)
```

Parameters

- **mapping_list** (*List[str]*) – Lookup list of tokens (e.g. words, characters).
- **label_from** (*LT*) – Attribute label of original instance (non-replaced).
- **label_to** (*LT*) – Attribute label of perturbed instance (replaced).
- **tokenizer** (*Callable, optional*) – Function to tokenize instance data (e.g. words, characters). Defaults to default_tokenizer.
- **detokenizer** (*Callable, optional*) – Function to detokenize tokens into instance data. Defaults to default_detokenizer.

classmethod from_nlpaug(*augmenter*, *label_from='original'*, *label_to='perturbed'*, ***augment_kwargs*)

Construct a *OneToOnePerturbation* from a nlpaug Augmenter.

Example

Add random spaces to words in a sentence using *nlpaug.augmenter.word.SplitAug()*:

```
>>> import nlpaug.augmenter.word as naw
>>> OneToOnePerturbation.from_nlpaug(naw.SplitAug(), label_to='with_extra_space
↪')
```

Or add keyboard typing mistakes to lowercase characters in a sentence using *nlpaug.augmenter.char.KeyboardAug()*:

```
>>> import nlpaug.augmenter.char as nac
>>> augmenter = nac.KeyboardAug(include_upper_case=False,
>>>                             include_special_char=False,
>>>                             include_numeric=False)
>>> OneToOnePerturbation.from_nlpaug(augmenter, label_from='no_typos', label_to=
↪'typos')
```

Parameters

- **augmenter** (*Augmenter*) – Class with *.augment()* function applying a perturbation to a string.
- **label_from** (*LT, optional*) – Attribute label of original instance. Defaults to 'original'.
- **label_to** (*LT, optional*) – Attribute label of perturbed instance. Defaults to 'perturbed'.
- ****augment_kwargs** – Optional arguments passed to *.augment()* function.

classmethod from_string(*prefix=None*, *suffix=None*, *replacement=None*, *label_from='original'*, *label_to='perturbed'*, *connector=' '*, *connector_before=None*, *connector_after=None*)

Construct a *OneToOnePerturbation* from a string (replacement, prefix and/or suffix).

Provides the ability to replace each instance string with a new one, add a prefix to each instance string and/or add a suffix to each instance string. At least one of *prefix*, *suffix* or *replacement* should be a string to apply the replacement.

Example

Add a random unrelated string 'Dit is ongerelateerd.' to each instance (as prefix), where you expect that predictions will not change:

```
>>> OneToOnePerturbation.from_string(prefix='Dit is ongerelateerd.', label_to=
↪'with_prefix')
```

Or add a negative string 'Dit is negatief!' to each instance (as suffix on the next line), where you expect that instances will have the same label or become more negative:

```
>>> OneToOnePerturbation.from_string(suffix='Dit is negatief!',
>>>                                  connector_after='\n',
>>>                                  label_to='more_negative')
```

Or replace all instances with 'UNKWRDZ': >>> OneToOnePerturbation.from_string(replacement='UNKWRDZ')

Raises

ValueError – At least one of *prefix*, *suffix* and *replacement* should be provided.

Parameters

- **label_from** (*LT*) – Attribute label of original instance. Defaults to 'original'.
- **label_to** (*LT*) – Attribute label of perturbed instance. Defaults to 'perturbed'.
- **prefix** (*Optional[str], optional*) – Text to add before *instance.data*. Defaults to None.
- **suffix** (*Optional[str], optional*) – Text to add after *instance.data*. Defaults to None.
- **replacement** (*Optional[str], optional*) – Text to replace *instance.data* with. Defaults to None.
- **connector** (*str*) – General connector between *prefix*, *instance.data* and *suffix*. Defaults to ' '.

- **connector_before** (*Optional[str], optional*) – Overrides connector between *prefix* and *instance.data*, if it is None *connector* is used. Defaults to None.
- **connector_after** (*Optional[str], optional*) – Overrides connector between *instance.data* and *suffix*, if it is None *connector* is used. Defaults to None.

classmethod from_tuples(*tuples*, *label_from*, *label_to*, *tokenizer=<function word_tokenizer>*, *detokenizer=<function word_detokenizer>*)

Construct a *OneToOnePerturbation* from tuples.

A function is constructed where if first aims to perform the mapping from the tokens on the left-hand side (LHS) to the right-hand side (RHS), and if this has no result it aims to perform the mapping from the tokens on the RHS to the LHS.

Example

For example, if *[('he', 'she')]* with *label_from='male'* and *label_to='female'* is provided it first checks whether the tokenized instance contains the word *'he'* (and if so applies the perturbation and returns), and otherwise aims to map *'she'* to *'he'*. If neither is possible, it returns None.

```
>>> tuples = [('he', 'she'),
>>.          ('his', 'her')]
>>> OneToOnePerturbation.from_tuples(tuples, label_from='male', label_to='female
↪')
```

Parameters

- **tuples** (*List[Tuple[str, str]]*) – Lookup tuples to map tokens (e.g. words, characters).
- **label_from** (*LT*) – Attribute label of original instance (left-hand side of tuples).
- **label_to** (*LT*) – Attribute label of perturbed instance (right-hand side of tuples).
- **tokenizer** (*Callable, optional*) – Function to tokenize instance data (e.g. words, characters). Defaults to default_tokenizer.
- **detokenizer** (*Callable, optional*) – Function to detokenize tokens into instance data. Defaults to default_detokenizer.

perturb(*instance*)

Apply a perturbation function to a single *TextInstance*, getting a single result per instance.

Parameters

- **perturbation_function** (*Callable*) – Perturbation function to apply, including attribute label of original instance and the resulting instance. Should return None if no perturbation has been applied.
- **instance** (*TextInstance*) –

Returns

None if no perturbation has been applied.
Otherwise a sequence of perturbed TextInstances, and attribute labels for the original and perturbed instances.

Return type
Optional[Sequence[Tuple[TextInstance, Sequence[Tuple[KT, LT]]]]]

class text_sensitivity.perturbation.base.**Perturbation**(*perturbation_function*)

Bases: Readable

Apply a perturbation function to a single *TextInstance*.

Parameters

perturbation_function (*Callable*) – Perturbation function to apply, including attribute label of original instance and resulting instance(s). Should return None if no perturbation has been applied.

classmethod from_dict(**args*, ***kwargs*)

Alias for *Perturbation.from_dictionary()*.

classmethod from_dictionary(**args*, ***kwargs*)

Construct a *Perturbation* from a dictionary.

Return type

Perturbation

classmethod from_function(*function*, *label_from='original'*, *label_to='perturbed'*)

Construct a *Perturbation* from a perturbation applied to a string.

Example

Make each sentence uppercase:

```
>>> OneToOnePerturbation(str.upper, 'not_upper', 'upper')
```

Parameters

- **function** (*Callable[[str], Optional[Union[str, Sequence[str]]]]*) – Function to apply to each string. Return None if no change was applied.
- **label_from** (*LT, optional*) – Attribute label of original instance. Defaults to 'original'.
- **label_to** (*LT, optional*) – Attribute label of perturbed instance. Defaults to 'perturbed'.

classmethod from_str(**args*, ***kwargs*)

Alias for *Perturbation.from_string()*.

classmethod from_string(**args*, ***kwargs*)

Construct a *Perturbation* from a string.

perturb(*instance*)

Apply perturbation to a single TextInstance.

Parameters

instance (*TextInstance*) –

text_sensitivity.perturbation.base.**as_list**(*x*)

Ensure an element *x* is a list.

Return type

list

text_sensitivity.perturbation.base.**format_identifier**(*instance*, *key*)

Format identifier of child.

text_sensitivity.perturbation.base.**one_to_many_dictionary_mapping**(*instance*, *dictionary*, *label_from*, *label_to*, *n*, *tokenizer*, *detokenizer*)

Create one-to-many replacement for a *TextInstance*.

Parameters

- **instance** (*TextInstance*) – Instance to create mapping for.
- **dictionary** (*Dict[str, List[str]]*) – Options for each token.
- **label_from** (*LT*) – Label of original element.
- **label_to** (*LT*) – Label of element with replacements applied.
- **n** (*int*) – Number of replacements for each instance.
- **tokenizer** (*Callable[[str], List[str]]*) – Tokenize string into sequence of tokens.
- **detokenizer** (*Callable[[List[str]], str]*) – Detokenize sequence of tokens to string.

Yields

Optional[List[Tuple[str, LT, LT]]] –

None if no change applied, or list of tuples containing detokenized instance, original label and replaced label.

Return type

Optional[List[Tuple[str, TypeVar(LT), TypeVar(LT)]]]

text_sensitivity.perturbation.base.**one_to_one_dictionary_mapping**(*instance*, *dictionary*, *label_from*, *label_to*, *tokenizer*, *detokenizer*)

Create one-to-one replacement for a *TextInstance*.

Parameters

- **instance** (*TextInstance*) – Instance to create mapping for.
- **dictionary** (*Dict[str, List[str]]*) – Options for each token.
- **label_from** (*LT*) – Label of original element.
- **label_to** (*LT*) – Label of element with replacements applied.
- **tokenizer** (*Callable[[str], List[str]]*) – Tokenize string into sequence of tokens.
- **detokenizer** (*Callable[[List[str]], str]*) – Detokenize sequence of tokens to string.

Yields

Optional[Tuple[str, LT, LT]] –

None if no change applied, or tuple containing detokenized instance, original label and replaced label.

Return type

Optional[Tuple[str, TypeVar(LT), TypeVar(LT)]]

text_sensitivity.perturbation.base.**oneway_dictionary_mapping**(*instance*, *dictionary*, *label_from*, *label_to*, *n*, *tokenizer*, *detokenizer*)

Create corresponding replacements for tokens in a *TextInstance*.

Parameters

- **instance** (*TextInstance*) – Instance to create mapping for.
- **dictionary** (*Dict[str, List[str]]*) – Options for each token.
- **label_from** (*LT*) – Label of original element.
- **label_to** (*LT*) – Label of element with replacements applied.
- **n** (*int*) – Number of replacements to pick.
- **tokenizer** (*Callable[[str], List[str]]*) – Tokenize string into sequence of tokens.
- **detokenizer** (*Callable[[List[str]], str]*) – Detokenize sequence of tokens to string.

Yields
Iterator[Optional[Tuple[str, LT, LT]]] – Detokenized instance, original label and replaced label.

Return type
Iterator[Optional[Tuple[str, TypeVar(LT), TypeVar(LT)]]]

text_sensitivity.perturbation.characters module

Create character-level perturbations (*text_sensitivity.perturbation.base.Perturbation*).

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.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

text_sensitivity.perturbation.sentences module

Create sentence-level perturbations (*text_sensitivity.perturbation.base.Perturbation*).

text_sensitivity.perturbation.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

text_sensitivity.perturbation.sentences.**to_lower**()

Make all characters in a string lowercase.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

text_sensitivity.perturbation.sentences.**to_upper**()

Make all characters in a string uppercase.

Returns
Object able to apply perturbations on strings or TextInstances.

Return type
Perturbation

text_sensitivity.perturbation.words module

Create word-level perturbations (*text_sensitivity.perturbation.base.Perturbation*).

5.3.3 text_sensitivity.ui

Submodules:

text_sensitivity.ui.notebook module

Extension of *genbase.ui.notebook* for custom rendering of `text_sensitivity.

class text_sensitivity.ui.notebook.**Render**(**configs*)

Bases: Render

Rendered for *text_sensitivity*.

custom_tab(*config*, ***renderargs*)

Custom tab, showing call arguments (*callargs*) for a sensitivity test.

Return type
str

Parameters
config (*dict*) –

property custom_tab_title

Title of custom tab.

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

Generic formatting for titles.

Return type
str

Parameters

- **title** (*str*) –
- **h** (*str*) –

get_renderer(*meta*)

Get rendererer depending on meta descriptor.

Parameters
meta (*dict*) –

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

Generic formatting for subtitles.

Return type
str

Parameters
meta (*dict*) –

property tab_title

Title of main tab.

text_sensitivity.ui.notebook.**h**(*title*)

Format title as HTML h3.

Return type
str

text_sensitivity.ui.notebook.**metrics_renderer**(*meta*, *content*, ***renderargs*)

Render *text_sensitivity.return_types.Metrics* as HTML.

Return type
str

Parameters

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

text_sensitivity.ui.notebook.**score_renderer**(*meta*, *content*, ***renderargs*)

Render score as HTML.

Return type
str

Parameters

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

text_sensitivity.ui.notebook.**success_test_renderer**(*meta*, *content*, ***renderargs*)

Render *text_sensitivity.return_types.SuccessTest* as HTML.

Return type
str

Parameters

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

Submodules:

5.3.4 text_sensitivity.metrics module

Module for sensitivity metrics.

class text_sensitivity.metrics.**FairnessMetrics**(*attributes*, *ground_truth*, *predictions*, *pos_label=None*, *priviliged_groups=None*, *unpriviliged_groups=None*)

Bases: object

Calculate fairness metrics for provided attributes, ground truth values and predictions.

Parameters

- **attributes** (*Dict[str, LabelProvider]*) – Dictionary of named attributes (e.g. sex, ethnicity, ...).
- **ground_truth** (*LabelProvider*) – Ground truth labels of instances.
- **predictions** (*LabelProvider*) – Predicted labels of instances.
- **pos_label** (*Optional[LT], optional*) – Positive label. Defaults to None.
- **priviliged_groups** (*Optional[List[Dict[str, LT]]], optional*) – Dictionary with the priviliged group names for some attributes. Defaults to None.
- **unpriviliged_groups** (*Optional[List[Dict[str, LT]]], optional*) – Dictionary with the unpriviliged group names for some attributes. Defaults to None.

Example

Calculate fairness metrics for *sex* (male/female) and *race* (Caucasian, Hispanic, African-American):

```
>>> attributes = {'sex': ..., 'race': ..., 'name': ...}
>>> FairnessMetric(attributes=attributes,
...                ground_truth=ground_truth,
...                predictions=predictions,
...                priviliged_groups={'sex': 'male', 'race': 'caucasian'})
```

property accuracy

$(TP + TN)/(P + N)$.

Type
Accuracy

property all

All indices.

property disparate_impact

Alias for statistical parity ratio.

property error_rate

$(FP + FN)/(TP + FP + FN + TN)$.

Type
Error rate

property false_discovery_rate

$FP/(TP + FP)$.

Type
False Discovery Rate (FDR)

property false_negative_rate

$FN/(FP + TP)$.

Type
False Negative Rate (FNR)

property false_omission_rate

$FN/(TN + FN)$.

Type
False Omission Rate (FOR)

property false_positive_rate

$FP/(TN + FN)$.

Type
False Positive Rate (FPR)

label_metrics(*keys*)

Calculate label metrics for given keys.

property mean_difference

Alias for statistical parity difference.

property negative_predictive_value

$TN/(TN + FN)$.

Type
Negative Predictive Value (NPV)

property positive_predicted_value

$TP/(TP + FP)$, alias for precision.

Type
Positive Predictive Value (NPV)

property precision

$TP/(TP + FP)$.

Type
Precision

property priviliged

Indices of priviliged instances.

property recall

$TP/(TP + FN)$

Type
Recall

property selection_rate

$(TP + FP)/(TP + FP + TN + FN)$.

Type
Selection rate

property sensitivity

Sensitivity, alias for *recall*.

property specificity

$TN/(FN + TN)$.

Type
Specificity

property statistical_parity

Statistical parity $Pr(Y = 1) = P/(P + N)$.

property true_negative_rate

True Negative Rate (TNR), alias for *specificity*.

property true_positive_rate

True Positive Rate (TPR), alias for *recall*.

property unpriviliged

Indices of unpriviliged instances.

class text_sensitivity.metrics.**Metric**(*name*, *all*, *priviliged*, *unpriviliged*, *abbr=None*)

Bases: Readable

Named metric.

Parameters

- **name** (*str*) – Name of metric.
- **all** (*float*) – Score of all
- **priviliged** (*float*) – Score of priviliged group (e.g. sex = male).
- **unpriviliged** (*float*) – Score for unpriviliged group (e.g. sex = not male).
- **abbr** (*Optional[str], optional*) – Abbreviation of name. Defaults to None.

property difference

Difference between unpriviliged and priviliged scores.

$$\text{metric}|A = \text{unprivileged} - \text{metric}|A = \text{privileged}$$

property ratio

Ratio between unpriviliged and priviliged scores.

$$\frac{\text{metric}|A = \text{unprivileged}}{\text{metric}|A = \text{privileged}}$$

text_sensitivity.metrics.**binarize**(*labels*, *select*, *new_label='priviliged'*, *other_label='unpriviliged'*)

Convert labels in a LabelProvider to binary labels *new_label* and *other_label*.

Parameters

- **labels** (*LabelProvider*) – LabelProvider to convert attribute labels to *new_label* and *other_label*.
- **select** (*Union[int, str, list, frozenset]*) – Label(s) to select.
- **new_label** (*str, optional*) – Label if attribute falls in selected values. Defaults to 'priviliged'.
- **other_label** (*str, optional*) – Label if attribute does not fall in selected values. Defaults to 'unpriviliged'.

Returns

Attribute labels with either *new_label* or *other_label*.

Return type

MemoryLabelProvider

5.3.5 text_sensitivity.return_types module

Return types for sensitivity (tests).

class text_sensitivity.return_types.**LabelMetrics**(*instances*, *label_metrics*, *type='sensitivity'*, *subtype='label_metrics'*, *callargs=None*, ***kwargs*)

Bases: Instances

Return type for labelwise metrics.

Parameters

- **instances** (*_type_*) – Instances.
- **label_metrics** (*_type_*) – Metric for each label.
- **type** (*Optional[str], optional*) – Type description. Defaults to 'sensitivity'.

- **subtype** (*Optional[str], optional*) – Subtype description. Defaults to 'label_metrics'.
- **callargs** (*Optional[dict], optional*) – Arguments used when the function was called. Defaults to None.

property content

Content as dictionary.

class text_sensitivity.return_types.**MeanScore**(*scores*, *label*, *instances*, *type='sensitivity'*, *subtype='mean_score'*, *callargs=None*, ***kwargs*)

Bases: Instances

Return type for *text_sensitivity.mean_score()*.

Parameters

- **scores** (*List[float]*) – Score for each instance.
- **label** (*str*) – Name of label.
- **instances** (*_type_*) – Instances.
- **type** (*Optional[str], optional*) – Type description. Defaults to 'sensitivity'.
- **subtype** (*Optional[str], optional*) – Subtype description. Defaults to 'mean_score'.
- **callargs** (*Optional[dict], optional*) – Arguments used when the function was called. Defaults to None.

property content

Content as dictionary.

class text_sensitivity.return_types.**SuccessTest**(*success_percentage*, *successes*, *failures*, *predictions=None*, *type='robustness'*, *subtype='input_space'*, *callargs=None*, ***kwargs*)

Bases: Instances

Return type for success test.

Parameters

- **success_percentage** (*float*) – Percentage of successful cases.
- **successes** (*_type_*) – Instances that succeeded.
- **failures** (*_type_*) – Instances that failed.
- **predictions** (*Optional[Union[LabelProvider, list, dict]], optional*) – Predictions to subdivide successes/ failures into labels. Defaults to None.
- **type** (*str, optional*) – Type description. Defaults to 'robustness'.
- **subtype** (*str, optional*) – Subtype description. Defaults to 'input_space'.
- **callargs** (*Optional[dict], optional*) – Arguments used when the function was called. Defaults to None.

property content

Content as dictionary.

5.3.6 text_sensitivity.sensitivity module

Sensitivity testing, for fairness and robustness.

text_sensitivity.sensitivity.**apply_perturbation**(*dataset*, *perturbation*)

Apply a perturbation to a dataset, getting the perturbed instances and corresponding attribute labels.

Examples

Repeat each string twice:

```
>>> from text_sensitivity.perturbation.sentences import repeat_k_times
>>> apply_perturbation(env, repeat_k_times(k=2))
```

Add the unrelated string 'This is unrelated.' before each instance:

```
>>> from text_sensitivity.perturbation import OneToOnePerturbation
>>> perturbation = OneToOnePerturbation.from_string(prefix='This is unrelated.')
>>> apply_perturbation(env, perturbation)
```

Parameters

- **dataset** (*Union[InstanceProvider, TextEnvironment]*) – Dataset to apply perturbation to (e.g. all data, train set, test set, set belonging to a given label, or subset of data for a (un)protected group).
- **perturbation** (Perturbation) – Perturbation to apply, one-to-one or one-to-many.

Returns

Perturbed instances and corresponding attribute labels.

Return type

Tuple[TextInstanceProvider, MemoryLabelProvider]

text_sensitivity.sensitivity.**compare_accuracy**(**args*, ***kwargs*)

Compare accuracy scores for each ground-truth label and attribute.

text_sensitivity.sensitivity.**compare_metric**(*env*, *model*, *perturbation*, ***kwargs*)

Get metrics for each ground-truth label and attribute.

Examples

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

```
>>> from text_sensitivity.perturbation.sentences import to_upper
>>> compare_metric(env, model, to_upper)
```

Compare metric when randomly adding 10 perturbed instances with typos to each instance in a dataset.

```
>>> from text_sensitivity.perturbation.characters import add_typos
>>> compare_metric(env, model, add_typos(n=10))
```

Parameters

- **env** (*TextEnvironment*) – Environment containing original instances (*.dataset*) and ground-truth labels (*.labels*).
- **model** (*AbstractClassifier*) – Black-box model to compare metrics on.
- **perturbation** (Perturbation) – Peturbation to apply.

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

Return type
LabelMetrics

text_sensitivity.sensitivity.**compare_precision**(**args*, ***kwargs*)

Compare precision scores for each ground-truth label and attribute.

text_sensitivity.sensitivity.**compare_recall**(**args*, ***kwargs*)

Compare recall scores for each ground-truth label and attribute.

text_sensitivity.sensitivity.**equal_ground_truth**(*ground_truth*, *instances*)

When you expect the ground-truth label will remain equal after the perturbation is applied.

Parameters

- **ground_truth** (*MemoryLabelProvider*) – Labelprovider.
- **instances** (*InstanceProvider*) – Instanceprovider.

Yields
Iterator[Tuple[KT, LT]] – Keys and corresponding labels.

Return type
Iterator[Tuple[TypeVar(KT), TypeVar(LT)]]

text_sensitivity.sensitivity.**input_space_robustness**(*model*, *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.

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 text_sensitivity.data.random.string import RandomAscii, RandomCyrillic,␣
↪RandomEmojis, RandomWhitespace
>>> input_space_robustness(model,
>>>                        [RandomWhitespace(), RandomAscii(),␣
↪RandomEmojis(base=True), RandomCyrillic('ru')],
>>>                        n_samples=1000,
>>>                        min_length=0,
>>>                        max_length=500)
```

Parameters

- **model** (*AbstractClassifier*) – Machine learning model to test.
- **generators** (*List[RandomString]*) – Random character generators.

- **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

text_sensitivity.sensitivity.**invariance**(*pattern*, *model*, *expectation=None*, ***kwargs*)

Test for the failure rate under invariance.

Parameters

- **pattern** (*str*) – String pattern to generate examples from.
- **model** (*AbstractClassifier*) – Model to test.
- **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

text_sensitivity.sensitivity.**mean_score**(*pattern*, *model*, *selected_label=None*, ***kwargs*)

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

Parameters

- **pattern** (*str*) –
- **model** (*AbstractClassifier*) – Model to generate scores from.
- **selected_label** (*Optional[LT], optional*) – Label name to select. If None is replaced by the first label. Defaults to None.

Returns
Mean score for the selected label, generated instances and label scores.

Return type
MeanScore

5.4 Changelog

All notable changes to `text_sensitivity` will be documented in this file.

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

5.4.1 Unreleased

Added

- Ensured full documentation

Changed

- Moved documentation to `sphinx`

Fixed

- Bugfix with lazy loading of locales

5.4.2 0.3.2 - 2022-03-21

Fixed

- Bugfix when Notebook UI

5.4.3 0.3.1 - 2022-03-16

Changed

- Custom sorting of metrics in return types
- Requires `genbase>=0.2.4`
- Requires `text_explainability>=0.6.1`
- Moved version information

5.4.4 0.3.0 - 2022-03-04

Added

- Fairness metrics

Changed

- Requires `genbase>=0.2.2`
- Requires `text_explainability>=0.6.0`
- Renamed `pyproject.toml` to `.portray` to avoid build errors
- Do not render `conf_mat` for model metrics

Fixed

- Added `person` provider to city generation in English

5.4.5 0.2.6 - 2021-12-06

Added

- Return type and notebook UI rendering for mean score
- Return type and notebook UI for invariance test

Changed

- Requires `genbase>=0.1.14`
- Requires `text_explainability>=0.5.8`

5.4.6 0.2.5 - 2021-12-02

Added

- Return types for sensitivity tests
- Rendering of sensitivity tests using `genbase.ui`

Changed

- Moved `SeedMixin` and `CaseMixin` to `genbase`
- Use `genbase.internationalization`
- Requires `genbase>=0.1.13`

5.4.7 0.2.4 - 2021-11-16

Fixed

- Bugfix in `OneToOnePerturbation.from_dictionary`
- Bugfix in `compare_metric`
- Bugfix in `one_to_one_dictionary_mapping`

5.4.8 0.2.3 - 2021-11-16

Fixed

- Bugfix in `oneway_dictionary_mapping`

5.4.9 0.2.2 - 2021-11-15

Fixed

- Bugfix in `one_to_one_dictionary_mapping`

5.4.10 0.2.1 - 2021-11-03

Added

- Invariance testing
- Comparison of mean scores (labelwise)

5.4.11 0.2.0 - 2021-10-08

Added

- Random license plate generation
- Added `SeedMixin` to `WordList`
- Added `CaseMixin` to `WordList`
- Robustness testing for random inputs
- Generate data from patterns
- Example usage for robustness testing and data generation

Changed

- Ability to generate items from `WordList`

5.4.12 0.1.10 - 2021-10-07

Added

- Perturbation imports (character, word, sentence) to `text_sensitivity.perturbation`
- Examples in README.md
- Attribute renaming in `text_sensitivity.data.random.entity`

Changed

- Updated usage with `text_explainability==0.5.0`
- Updated usage with `faker==8.16.0`

5.4.13 0.1.9 - 2021-10-02

Fixed

- Bugfix in reading .csv files

5.4.14 0.1.8 - 2021-10-02

Removed

- Removed cities from wordlists

5.4.15 0.1.7 - 2021-10-02

Added

- MANIFEST.in
- Security tests with bandit
- Ability to make random entities lowercase, uppercase or sentencecase
- Tests for `text_sensitivity.data.random.string`
- Tests for `text_sensitivity.data.random.entity`
- Additional documentation
- Ability to generate addresses/cities in a country with a likelihood based on their population

Removed

- Removed countries from wordlists

Fixed

- Bugfixes in `OneToOnePerturbation` and `OneToManyPerturbation`

5.4.16 0.1.6 - 2021-10-02

Changed

- Moved random string data generation from `text_sensitivity.data.random` to `text_sensitivity.data.random.string`
- Renamed `RandomData` to `RandomString`
- Seed behavior generalized in `SeedMixin`, only requiring a `self._seed` and `self._original_seed` to work with a class

Added

- Random multilingual entity generation with Python package `faker`
- Documentation and example usages for random entity generation

5.4.17 0.1.5 - 2021-10-01

Added

- Internationalization support
- Name of countries by language word list
- Top 100 most populous cities by country word list

5.4.18 0.1.4 - 2021-09-30

Added

- Citation information
- Documentation styling
- Generation of random Cyrillic text

5.4.19 0.1.3 - 2021-09-27

Added

- Documentation
- Ability to make `OneToOnePerturbation` from unordered list
- Extended one-to-one and one-to-many dictionary mappings

5.4.20 0.1.2 - 2021-09-24

Changed

- Proper n-times application of function with `OneToManyPerturbation`

Fixed

- Bugfix in character generation

5.4.21 0.1.1 - 2021-09-24

Added

- Example usage
- Sensitivity testing wrapper functions (compare accuracy, precision, recall)

5.4.22 0.1.0 - 2021-09-24

Added

- Random data generation
- One to one perturbation
- One to many perturbation
- Example perturbation functions
- README.md
- LICENSE
- CI/CD pipeline for flake8 testing
- setup.py

5.5 Contributing

...

5.5.1 Maintenance

Contributors

- Marcel Robeer (`@m.j.robeer`)
- Elize Herrewijnen (`@e.herrewijnen`)

Todo

Tasks yet to be done:

- Word-level perturbations
- Add fairness-specific metrics:
  - Counterfactual fairness
- Add expected behavior
  - Robustness: equal to prior prediction, or in some cases might expect that it deviates
  - Fairness: may deviate from original prediction
- Tests
  - Add tests for perturbations
  - Add tests for sensitivity testing schemes
- Add visualization ability

5.6 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

G

H

I

L

M

N

O

P

R

S

T

U

W