ferret Documentation

Release 0.3.5

Giuseppe Attanasio

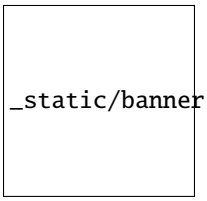
Feb 05, 2023

CONTENTS:

CHAPTER

ONE

FERRET



A python package for benchmarking interpretability techniques.

- Free software: MIT license
- Documentation: https://ferret.readthedocs.io.

```python
from transformers import AutoModelForSequenceClassification, AutoTokenizer
from ferret import Benchmark

name = "cardiffnlp/twitter-xlm-roberta-base-sentiment"
model = AutoModelForSequenceClassification.from_pretrained(name)
tokenizer = AutoTokenizer.from_pretrained(name)

bench = Benchmark(model, tokenizer)
explanations = bench.explain("You look stunning!", target=1)
evaluations = bench.evaluate_explanations(explanations, target=1)

bench.show_evaluation_table(evaluations)
```

1.1 Features

ferret offers a *painless* integration with Hugging Face models and naming conventions. If you are already using the transformers library, you immediately get access to our **Explanation and Evaluation API**.

1.1.1 Supported Post-hoc Explainers

- Gradient (plain gradients or multiplied by input token embeddings) (Simonyan et al., 2014)
- Integrated Gradient (plain gradients or multiplied by input token embeddings) (Sundararajan et al., 2017)
- SHAP (via Partition SHAP approximation of Shapley values) (Lundberg and Lee, 2017)
- LIME (Ribeiro et al., 2016)

1.1.2 Supported Evaluation Metrics

Faithfulness measures:

- AOPC Comprehensiveness (DeYoung et al., 2020)
- AOPC Sufficiency (DeYoung et al., 2020)
- Kendall's Tau correlation with Leave-One-Out token removal. (Jain and Wallace, 2019)

Plausibility measures:

- Area-Under-Precision-Recall-Curve (soft score) (DeYoung et al., 2020)
- Token F1 (hard score) (DeYoung et al., 2020)
- Token Intersection Over Union (hard score) (DeYoung et al., 2020)

See our paper for details.

1.2 Visualization

The *Benchmark* class exposes easy-to-use table visualization methods (e.g., within Jupyter Notebooks)

```
bench = Benchmark(model, tokenizer)

# Pretty-print feature attribution scores by all supported explainers
explanations = bench.explain("You look stunning!")
bench.show_table(explanations)

# Pretty-print all the supported evaluation metrics
evaluations = bench.evaluate_explanations(explanations)
bench.show_evaluation_table(evaluations)
```

1.3 Dataset Evaluations

The *Benchmark* class has a handy method to compute and average our evaluation metrics across multiple samples from a dataset.

```
import numpy as np
bench = Benchmark(model, tokenizer)

# Compute and average evaluation scores one of the supported dataset
samples = np.arange(20)
```

(continues on next page)

(continued from previous page)

```
hatexdata = bench.load_dataset("hatexplain")
sample_evaluations =  bench.evaluate_samples(hatexdata, samples)

# Pretty-print the results
bench.show_samples_evaluation_table(sample_evaluations)
```

1.4 Credits

This package was created with Cookiecutter and the *audreyr/cookiecutter-pypackage* project template.

- Cookiecutter: https://github.com/audreyr/cookiecutter
- *audreyr/cookiecutter-pypackage*: https://github.com/audreyr/cookiecutter-pypackage

Logo and graphical assets made by Luca Attanasio.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install ferret, run this command in your terminal:

```
$ pip install -U ferret-xai
```

This is the preferred method to install ferret, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ferret can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/g8a9/ferret
```

Or download the tarball:

```
$ curl -OJL https://github.com/g8a9/ferret/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use ferret in a project:

```
import ferret
```

CHAPTER

FOUR

MODULES

4.1 Benchmark

class ferret.benchmark.**Benchmark**(*model*, *tokenizer*, *explainers: Optional[List] = None*, *evaluators: Optional[List] = None*, *class_based_evaluators: Optional[List] = None*)

Generic interface to compute multiple explanations.

evaluate_explanation(*explanation: Union[Explanation, ExplanationWithRationale]*, *target*, *human_rationale=None*, *class_explanation: Optional[List[Union[Explanation, ExplanationWithRationale]]] = None*, *progress_bar=True*, ***evaluation_args*) → ExplanationEvaluation

explanation: Explanation to evaluate. target: target class for which we evaluate the explanation human rationale: List in one-hot-encoding indicating if the token is in the rationale (1) or not (i) class_explanation: list of explanations. The explanation in position 'i' is computed using as target class the class 'i'.

len = #target classes. If available, class-based scores are computed

evaluate_explanations(*explanations: List[Union[Explanation, ExplanationWithRationale]]*, *target*, *human_rationale=None*, *class_explanations=None*, *progress_bar=True*, ***evaluation_args*) → List[ExplanationEvaluation]

evaluate_samples(*dataset: BaseDataset*, *sample: Union[int, List[int]]*, *target=None*, *show_progress_bar: bool = True*, *n_workers: int = 1*, ***evaluation_args*) → Dict

Explain a dataset sample, evaluate explanations, and compute average scores.

explain(*text*, *target=1*, *progress_bar: bool = True*) → List[Explanation]

Compute explanations.

get_dataframe(*explanations*) → DataFrame

load_dataset(*dataset_name: str*, ***kwargs*)

score(*text*, *return_dict: bool = True*)

show_evaluation_table(*explanation_evaluations: List[ExplanationEvaluation]*, *apply_style: bool = True*) → DataFrame

Format explanations and evaluations scores into a colored table.

show_samples_evaluation_table(*evaluation_scores_by_explainer*, *apply_style: bool = True*) → DataFrame

Format dataset average evaluations scores into a colored table.

show_table(*explanations*, *apply_style: bool = True*, *remove_first_last: bool = True*) → DataFrame

Format explanations scores into a colored table.

style_evaluation(*table*)

4.2 Explainers

class ferret.explainers.gradient.**GradientExplainer**(*model*, *tokenizer*, *multiply_by_inputs: bool = True*)

NAME = 'Gradient'

compute_feature_importance(*text: str*, *target: False*, ***explainer_args*)

class ferret.explainers.gradient.**IntegratedGradientExplainer**(*model*, *tokenizer*, *multiply_by_inputs: bool = True*)

NAME = 'Integrated Gradient'

compute_feature_importance(*text*, *target*, ***explainer_args*)

class ferret.explainers.shap.**SHAPExplainer**(*model*, *tokenizer*)

NAME = 'Partition SHAP'

compute_feature_importance(*text*, *target=1*, ***explainer_args*)

class ferret.explainers.lime.**LIMEExplainer**(*model*, *tokenizer*)

NAME = 'LIME'

compute_feature_importance(*text*, *target=1*, ***explainer_args*)

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/g8a9/ferret/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

ferret could always use more documentation, whether as part of the official ferret docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/g8a9/ferret/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *ferret* for local development.

1. Fork the *ferret* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ferret.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ferret
$ cd ferret/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ferret tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/g8a9/ferret/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_ferret
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Giuseppe Attanasio <giuseppeattanasio6@gmail.com>
- Eliana Pastor <eliana.pastor@centai.eu>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2022-05-30)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX