

Conducting Research Experiments with QuaPy

Quantification: Predicting Class Frequencies
via Supervised Learning

Alejandro Moreo, Fabrizio Sebastiani

ISTI-CNR, Pisa, Italy
{alejandro.moreo,fabrizio.sebastiani}@isti.cnr.it

ESSAI
Athens, Greece – July 22-26, 2024



Introduction

- **QUAPY** is a Python-based, open-source software library for quantification
- Installation:

```

! pip install quapy

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting quapy
  Downloading Quapy-0.1.7-py3-none-any.whl (99 kB)
    _____ 99.1/99.1 KB 2.8 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from quapy) (1.3.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from quapy) (4.64.1)
Requirement already satisfied: xlrd in /usr/local/lib/python3.8/dist-packages (from quapy) (1.2.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from quapy) (1.2.0)
Collecting abstention
  Downloading abstention-0.1.3.1.tar.gz (24 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from quapy) (1
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from quapy) (3.2
Requirement already satisfied: numpy>=1.9 in /usr/local/lib/python3.8/dist-packages (from abstention-
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from abstention-
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from s
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from m
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!2.1.6,>=2.0.1 in /usr/local/lib/python3.8/c
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matp
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotli
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->c
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from pandas-dateu
Building wheels for collected packages: abstention
  Building wheel for abstention (setup.py) ... done
  Created wheel for abstention: filename=abstention-0.1.3.1-py3-none-any.whl size=25454 sha256=1b42c5
  Stored in directory: /root/.cache/pip/wheels/a0/dc/ed/2ddd390de2a6b43aa30f75155b50dfea2bc3b8ed9a5cfc
Successfully built abstention
Installing collected packages: abstention, quapy
Successfully installed abstention-0.1.3.1 quapy-0.1.7

```

GitHub repository

- **QUAPY** v0.1.8 available at <https://github.com/HLT-ISTI/QuaPy>

The screenshot displays the GitHub repository page for **HLT-ISTI / QuaPy**. The repository is described as "A framework for Quantification written in Python".

Your master branch isn't protected
Protect this branch from force pushing or deletion, or require status checks before merging. [View](#) [Protect this branch](#)

Files

File Name	Last Commit	Age
github-actions	Create pyproject.toml	last year
docs	adding template to docs	5 months ago
examples	custom practical example added	5 months ago
logs	adding logs	5 months ago
setup	update setup.py	2 months ago
github.com	update github.com	5 months ago
CHANGE_LOG.md	bring this and documenting	5 months ago
LICENSE	license updated	4 years ago
README.md	adding logos	5 months ago
TODO.md	Added NAL, INAC	9 months ago
prepare_template.sh	clearing	4 years ago
setup.py	Update setup.py	2 months ago
test-part-quantification-test-patch	many negative methods added	4 years ago

README **BSD-3-Clause license**

QuaPy

QuaPy is an open source framework for quantification (i.e. supervised prevalence estimation, or learning to quantify) written in Python.

QuaPy is based on the concept of "data sampler", and provides implementations of the most important aspects of the quantification workflow, such as (baseline and advanced) quantification methods, quantification-oriented model selection mechanisms, evaluation measures, and evaluation protocols used for evaluating quantification methods. QuaPy also makes available commonly used datasets, and offers visualization tools for facilitating the analysis and interpretation of the experimental results.

Last updates:

- Version 0.1.8 is released! major changes can be consulted [here](#).
- A detailed wiki is available [here](#).
- The developer API documentation is available [here](#).

Installation

```
pip install quapy
```

Releases

- QuaPy v0.1.8 (latest) (4 months ago)
- 2 releases

Packages

No packages published

[Publish your first package](#)

Contributors

4 contributors

Deployments

- github-pages 2 months ago
- 41 deployments

Languages

- Python 99.0%
- Shell 1.0%

GitHub repository: Examples

HLT-ISTI / QuaPy

Search: Type to search

Code Issues 1 Pull requests 1 Discussions Actions Projects Wiki Security Insights Settings

Files

master + Q

Go to file t

- .github
- docs
- examples
 - custom_quantifier.py
 - explicit_loss_minimization.py
 - lequa2022_experiments.py
 - lequa2022_experiments_recali...
 - model_selection.py
 - one_vs_all.py
 - quanet_example.py
 - uci_experiments.py
- quapy
 - .gitignore
 - LICENSE
 - README.md
 - SoBigData.png
 - TODO.txt
 - prepare_svmperf.sh
 - setup.py
 - svm-perf-quantification-ext.patch

QuaPy / examples /

Add file ...

AlexMoreo import fix in uci_experiments.py 1efe13c · 6 months ago History

Name	Last commit message	Last commit date
..		
custom_quantifier.py	import fix	7 months ago
explicit_loss_minimization.py	evaluation updated	7 months ago
lequa2022_experiments.py	import fix	7 months ago
lequa2022_experiments_recalib.py	fixing hyperparameters with prefixes, and replacing learner with c...	8 months ago
model_selection.py	preparing to merge	7 months ago
one_vs_all.py	evaluation updated	7 months ago
quanet_example.py	some bug fixes here and there	7 months ago
uci_experiments.py	import fix in uci_experiments.py	6 months ago

4 / 24

Online API documentation

quapy.functional

quapy.functional.HellingerDistance(*P*, *Q*)

Computes the Hellinger Distance (HD) between (discretized) distributions *P* and *Q*. The HD for two discrete distributions of *k* bins is defined as:

$$HD(P, Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^k (\sqrt{p_i} - \sqrt{q_i})^2}$$

- Parameters:**
- P** – real-valued array-like of shape (*k*,) representing a discrete distribution
 - Q** – real-valued array-like of shape (*k*,) representing a discrete distribution
- Returns:** float

quapy.functional.TopsoeDistance(*P*, *Q*, *epsilon*=1e-20)

Topsoe distance between two (discretized) distributions *P* and *Q*. The Topsoe distance for two discrete distributions of *k* bins is defined as:

$$Topsoe(P, Q) = \sum_{i=1}^k \left(p_i \log \left(\frac{2p_i + \epsilon}{p_i + q_i + \epsilon} \right) + q_i \log \left(\frac{2q_i + \epsilon}{p_i + q_i + \epsilon} \right) \right)$$

- Parameters:**
- P** – real-valued array-like of shape (*k*,) representing a discrete distribution
 - Q** – real-valued array-like of shape (*k*,) representing a discrete distribution
- Returns:** float

quapy.functional.adjusted_quantification(*prevalence_estim*, *tpr*, *fpr*, *clip*=True)

Implements the adjustment of ACC and PACC for the binary case. The adjustment for a prevalence estimate of the positive class *p* comes down to computing:

$$ACC(p) = \frac{p - fpr}{tpr - fpr}$$

- Parameters:**
- prevalence_estim** – float, the estimated value for the positive class
 - tpr** – float, the true positive rate of the classifier
 - fpr** – float, the false positive rate of the classifier
 - clip** – set to True (default) to clip values that might exceed the range [0,1]
- Returns:** float, the adjusted count



Online API documentation

quapy.data.datasets

`quapy.data.datasets.fetch_UCIDataset(dataset_name, data_home=None, test_split=0.3, verbose=False) → Dataset`

Loads a UCI dataset as an instance of `quapy.data.base.Dataset`, as used in [Pérez-Gállego, P., Quevedo, J. R., & del Coz, J. J. \(2017\). Using ensembles for problems with characterizable changes in data distribution: A case study on quantification. *Information Fusion*, 34, 87-100.](#) and [Pérez-Gállego, P., Castano, A., Quevedo, J. R., & del Coz, J. J. \(2019\). Dynamic ensemble selection for quantification tasks. *Information Fusion*, 45, 1-15.](#) The datasets do not come with a predefined train-test split (see `fetch_UCILabelledCollection()` for further information on how to use these collections), and so a train-test split is generated at desired proportion. The list of valid dataset names can be accessed in `quapy.data.datasets.UCI_DATASETS`

Parameters:

- `dataset_name` – a dataset name
- `data_home` – specify the quapy home directory where collections will be dumped (leave empty to use the default `~/quay_data/` directory)
- `test_split` – proportion of documents to be included in the test set. The rest conforms the training set
- `verbose` – set to True (default is False) to get information (from the UCI ML repository) about the datasets

Returns: a `quapy.data.base.Dataset` instance

`quapy.data.datasets.fetch_UCILabelledCollection(dataset_name, data_home=None, verbose=False)`
– `Dataset`

Loads a UCI collection as an instance of `quapy.data.base.LabelledCollection`, as used in [Pérez-Gállego, P., Quevedo, J. R., & del Coz, J. J. \(2017\). Using ensembles for problems with characterizable changes in data distribution: A case study on quantification. *Information Fusion*, 34, 87-100.](#) and [Pérez-Gállego, P., Castano, A., Quevedo, J. R., & del Coz, J. J. \(2019\). Dynamic ensemble selection for quantification tasks. *Information Fusion*, 45, 1-15.](#) The datasets do not come with a predefined train-test split, and so Pérez-Gállego et al. adopted a 5FCVx2 evaluation protocol, meaning that each collection was used to generate two rounds (hence the x2) of 5 fold cross validation. This can be reproduced by using `quapy.data.base.Dataset.kFCV()`, e.g.:



```
>>> import quapy as qp
>>> collection = qp.datasets.fetch_UCILabelledCollection("yeast")
>>> for data in qp.data.Dataset.kFCV(collection, nfoldd=5, nrepeats=2):
>>>     ...
```


The list of valid dataset names can be accessed in `quapy.data.datasets.UCI_DATASETS`

Parameters:

- `dataset_name` – a dataset name
- `data_home` – specify the quapy home directory where collections will be dumped (leave empty to use the default `~/quay_data/` directory)

Online Wiki



 HLT-ISTI / QuaPy

>
+
🔄
🔍
📧


[Code](#)
[Issues](#) ¹
[Pull requests](#) ¹
[Discussions](#)
[Actions](#)
[Projects](#)
[Wiki](#)
[Security](#)
[Insights](#)
[Settings](#)

Home

[Edit](#) [New page](#)

Alejandro Moreo edited this page on Feb 14 - 3 revisions

Welcome to the QuaPy wiki!

In this wiki we show examples that illustrate the main concepts behind QuaPy. This wiki includes:

- [Datasets](#)
- [Evaluation](#)
- [Protocols](#)
- [Methods](#)
- [SVMperf](#)
- [Model Selection](#)
- [Plotting](#)

+ Add a custom footer

Pages

[Home](#)

▸ [Datasets](#)

▸ [Evaluation](#)

▸ [ExplicitLossMinimization](#)

▸ [Methods](#)


▸ [Model Selection](#)



▸ [Plotting](#)


▸ [Protocols](#)

+ Add a custom sidebar

Clone this wiki locally

<https://github.com/HLT-ISTI/Qual> 



 HLT-ISTI / QuaPy

>
+
🔄
🔗
📧


<
Code
Issues
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
...

Datasets

[Edit](#)
[New page](#)

Alejandro Moreo edited this page on Feb 14 · 8 revisions

Datasets

QuaPy makes available several datasets that have been used in quantification literature, as well as an interface to allow anyone import their custom datasets.

A *Dataset* object in QuaPy is roughly a pair of *LabelledCollection* objects, one playing the role of the training set, another the test set. *LabelledCollection* is a data class consisting of the (iterable) instances and labels. This class handles most of the sampling functionality in QuaPy. Take a look at the following code:

```
import quapy as qp
import quapy.functional as F

instances = [
    '1st positive document', '2nd positive document',
    'the only negative document',
    '1st neutral document', '2nd neutral document', '3rd neutral document'
]
labels = [2, 2, 0, 1, 1, 1]

data = qp.data.LabelledCollection(instances, labels)
print(F.strprev(data.prevalence(), prec=2))
```

Output the class prevalences (showing 2 digit precision):

```
[0.17, 0.50, 0.33]
```

One can easily produce new samples at desired class prevalence values:

```
sample_size = 10
```

Pages 8

▶ [Home](#)

▶ Datasets

Datasets

Reviews Datasets

Twitter Sentiment Datasets

UCI Machine Learning

Issues:

LeQua Datasets

Adding Custom Datasets

Data Processing

▶ [Evaluation](#)

▶ [ExplicitLossMinimization](#)

▶ [Methods](#)

▶ [Model Selection](#)

▶ [Plotting](#)

▶ [Protocols](#)



Online Wiki

Meta Models

By *meta* models we mean quantification methods that are defined on top of other quantification methods, and that thus do not squarely belong to the aggregative nor the non-aggregative group (indeed, *meta* models could use quantifiers from any of those groups). *Meta* models are implemented in the `qp.method.meta` module.

Ensembles

QuaPy implements (some of) the variants proposed in:

- Pérez-Gállego, P., Quevedo, J. R., & del Coz, J. J. (2017). *Using ensembles for problems with characterizable changes in data distribution: A case study on quantification*. *Information Fusion*, 34, 87-100.
- Pérez-Gállego, P., Castano, A., Quevedo, J. R., & del Coz, J. J. (2019). *Dynamic ensemble selection for quantification tasks*. *Information Fusion*, 45, 1-15.

The following code shows how to instantiate an Ensemble of 30 *Adjusted Classify & Count* (ACC) quantifiers operating with a *Logistic Regressor* (LR) as the base classifier, and using the *average* as the aggregation policy (see the original article for further details). The last parameter indicates to use all processors for parallelization.

```
import quapy as qp
from quapy.method.aggregative import ACC
from quapy.method.meta import Ensemble
from sklearn.linear_model import LogisticRegression

dataset = qp.datasets.fetch_UCIDataset('haberman')

model = Ensemble(quantifier=ACC(LogisticRegression()), size=30, policy='ave', n_jobs=-1)
model.fit(dataset.training)
estim_prevalence = model.quantify(dataset.test.instances)
```

Other aggregation policies implemented in QuaPy include:

- 'ptr' for applying a dynamic selection based on the training prevalence of the ensemble's members
- 'ds' for applying a dynamic selection based on the Hellinger Distance
- *any valid quantification measure* (e.g., 'mse') for performing a static selection based on the performance estimated for each member of the ensemble in terms of that evaluation metric.

Features

- The most important features of QUAPY include:
 - QUAPY provides implementations of many **methods** (basic and advanced) implementing the same interface
 - QUAPY implements standard **evaluation protocols**
 - Provides implementations of the most important **evaluation metrics**
 - Access to many popular **datasets** for quantification
 - QUAPY provides native support to **multiclass** quantification
 - Automatic optimization of hyperparameters (**model selection**)
 - **Visualization** tools



Example: binary quantification

```
from sklearn.linear_model import LogisticRegression

dataset = qp.datasets.fetch_reviews('kindle', tfidf=True, min_df=5)
training, test = dataset.train_test

# create an "Adjusted Classify & Count" quantifier
model = qp.method.aggregative.ACC(LogisticRegression())
model.fit(training)

estim_prevalence = model.quantify(test.instances)
true_prevalence = test.prevalence()

error = qp.error.mae(true_prevalence, estim_prevalence)

print(f'Mean Absolute Error (MAE)={error:.3f}')
```

```
loading /root/quapy_data/reviews/kindle/train.txt: 100%|██████████| 3821/3821 [00:00<00:00, 316371.59it/s]
loading /root/quapy_data/reviews/kindle/test.txt: 100%|██████████| 21591/21591 [00:00<00:00, 562546.00it/s]
Mean Absolute Error (MAE)=0.006
```

Example: multiclass quantification

```
dataset = qp.datasets.fetch_twitter('hcr')
training, test = dataset.train_test

# create an "Adjusted Classify & Count" quantifier
model = qp.method.aggregative.ACC(LogisticRegression())
model.fit(training)

estim_prevalence = model.quantify(test.instances)
true_prevalence = test.prevalence()

error = qp.error.mae(true_prevalence, estim_prevalence)

print(f'Mean Absolute Error (MAE)={error:.3f}')
```

```
Downloading https://zenodo.org/record/4255764/files/tweet\_sentiment\_quantification\_snam.zip
downloaded 238.17 MB / 238.17 MB
loading /root/quapy_data/tweet_sentiment_quantification_snam/train/hcr.train+dev.feature.txt: 100%
|-- filling matrix of shape (1594, 222046): 100%|██████████| 1594/1594 [00:01<00:00, 1040.91it/s]
loading /root/quapy_data/tweet_sentiment_quantification_snam/test/hcr.test.feature.txt: 100%|██████████|
|-- filling matrix of shape (798, 222046): 100%|██████████| 798/798 [00:00<00:00, 1908.64it/s]
Mean Absolute Error (MAE)=0.038
```

Evaluation Protocols

- QUAPY implements most important evaluation protocols (example: APP)
- Results can be stored as a pandas' dataframe

```
# evaluation measures typically used in quantification literature
from quapy.protocol import UPP
import quapy.functional as F
import pandas as pd

qp.environ['SAMPLE_SIZE'] = 100

report = qp.evaluation.evaluation_report(model, protocol=UPP(test), error_metrics=['mae', 'mrae', 'kld', 'nkld'])

pd.set_option('display.expand_frame_repr', False)
report['estim-prev'] = report['estim-prev'].map(F.strprev)
print(report)

print('Averaged values:')
print(f"MAE={report['mae'].mean():.3f}")
print(f"MRAE={report['mrae'].mean():.3f}")
print(f"KLD={report['kld'].mean():.3f}")
print(f"NKLD={report['nkld'].mean():.3f}")
```

	true-prev		estim-prev	mae	mrae	kld	nkld
0	[0.44, 0.12, 0.0, 0.05, 0.04, 0.07, 0.28]		[0.432, 0.116, 0.000, 0.062, 0.053, 0.113, 0.224]	0.019464	0.189856	0.018309	0.009154
1	[0.39, 0.05, 0.09, 0.26, 0.1, 0.07, 0.04]		[0.429, 0.048, 0.000, 0.234, 0.239, 0.050, 0.000]	0.050600	0.520629	0.305042	0.151349
2	[0.02, 0.05, 0.02, 0.48, 0.27, 0.09, 0.07]		[0.081, 0.049, 0.000, 0.506, 0.185, 0.102, 0.078]	0.030468	0.550649	0.068955	0.034464
3	[0.46, 0.32, 0.0, 0.02, 0.07, 0.11, 0.02]		[0.478, 0.320, 0.000, 0.006, 0.032, 0.150, 0.021]	0.014838	0.212554	0.027286	0.013642
4	[0.12, 0.03, 0.27, 0.1, 0.12, 0.3, 0.06]		[0.138, 0.030, 0.246, 0.102, 0.100, 0.300, 0.005]	0.022825	0.226465	0.082289	0.041121
...
95	[0.04, 0.14, 0.13, 0.09, 0.52, 0.03, 0.05]		[0.013, 0.139, 0.157, 0.099, 0.574, 0.018, 0.000]	0.025756	0.324492	0.104496	0.052201
96	[0.16, 0.0, 0.029, 0.21, 0.23, 0.02, 0.09]		[0.163, 0.000, 0.272, 0.168, 0.272, 0.079, 0.045]	0.029729	0.470370	0.053128	0.026558
97	[0.04, 0.02, 0.02, 0.17, 0.19, 0.26, 0.3]		[0.030, 0.020, 0.049, 0.152, 0.142, 0.196, 0.411]	0.039911	0.333049	0.044520	0.022256
98	[0.06, 0.47, 0.14, 0.07, 0.16, 0.01, 0.09]		[0.000, 0.471, 0.163, 0.072, 0.263, 0.014, 0.016]	0.038178	0.401266	0.200046	0.099691
99	[0.3, 0.06, 0.02, 0.09, 0.51, 0.02, 0.0]		[0.335, 0.057, 0.022, 0.110, 0.458, 0.019, 0.000]	0.016266	0.087341	0.006745	0.003372

```
[100 rows x 6 columns]
Averaged values:
MAE=0.032
MRAE=0.414
KLD=0.089
NKLD=0.044
```

Model Selection

```

import quapy as qp
from quapy.protocol import UPP
from quapy.method.aggregative import DistributionMatchingY
from sklearn.linear_model import LogisticRegression
import numpy as np

model = DistributionMatchingY(LogisticRegression())

qp.environ['SAMPLE_SIZE'] = 100
qp.environ['N_JOBS'] = -1 # explore hyper-parameters in parallel

training, test = qp.datasets.fetch_UCIMulticlassDataset('digits').train_test

model.fit(training)
mae_score = qp.evaluation.evaluate(model, protocol=UPP(test), error_metric='mae')
print(f'MAE (not optimized)={mae_score:.5f}')

training, validation = training.split_stratified(train_prop=0.7)
protocol = UPP(validation, repeats=100)

param_grid = {
    'classifier_C': np.logspace(-3,3,7),
    'nbins': [4, 8, 10, 16],
}

model = qp.model_selection.GridSearchQ(
    model=model,
    param_grid=param_grid,
    protocol=protocol,
    error='mae', # the error to optimize is the MAE (a quantification-oriented loss)
    refit=True, # retrain on the whole labelled set once done
    verbose=True, # show information as the process goes on
).fit(training)

print(f'model selection ended: best hyper-parameters={model.best_params_}')
model = model.best_model_

# evaluation in terms of MAE
# we use the same evaluation protocol (APP) on the test set
mae_score = qp.evaluation.evaluate(model, protocol=UPP(test), error_metric='mae')

print(f'MAE (optimized)={mae_score:.5f}')

```

Comparing different models: Plots

```

from quapy.method.aggregative import CC, ACC, PCC, PACC, EMQ, DistributionMatching
from sklearn.calibration import CalibratedClassifierCV
from tqdm import tqdm

def base_classifier():
    return LogisticRegression(class_weight='balanced')

def models():
    yield 'CC', CC(base_classifier())
    yield 'ACC', ACC(base_classifier())
    yield 'PCC', PCC(base_classifier())
    yield 'PACC', PACC(base_classifier())
    yield 'EMQ', EMQ(CalibratedClassifierCV(base_classifier()))
    yield 'HDy', DistributionMatching(base_classifier())

training, test = qp.datasets.fetch_reviews('kindle', tfidf=True, min_df=5).train_test

method_names, true_prevs, estim_prevs, tr_prevs = [], [], [], []

for method_name, model in tqdm(models(), 'generating results', total=6):
    model.fit(training)
    true_prev, estim_prev = qp.evaluation.prediction(model, APP(test, repeats=100, random_state=0))

    method_names.append(method_name)
    true_prevs.append(true_prev)
    estim_prevs.append(estim_prev)
    tr_prevs.append(training.prevalence())

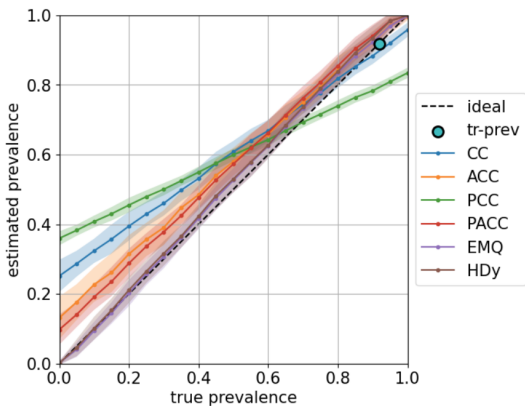
loading /root/quapy_data/reviews/kindle/train.txt: 100%|██████████| 3821/3821 [00:00<00:00, 253671.14it/s]
loading /root/quapy_data/reviews/kindle/test.txt: 100%|██████████| 21591/21591 [00:00<00:00, 278491.22it/s]
generating results: 100%|██████████| 6/6 [00:15<00:00, 2.63s/it]

```

Comparing different models: Plots

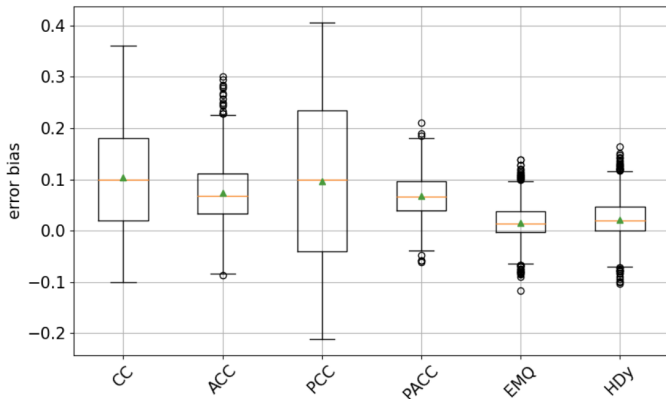
```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 6]
plt.rcParams['figure.dpi'] = 100
plt.rcParams['font.size'] = 15
```

```
qp.plot.binary_diagonal(method_names, true_prevs, estim_prevs, train_prev=tr_prevs[0])
```



Comparing different models: Plots

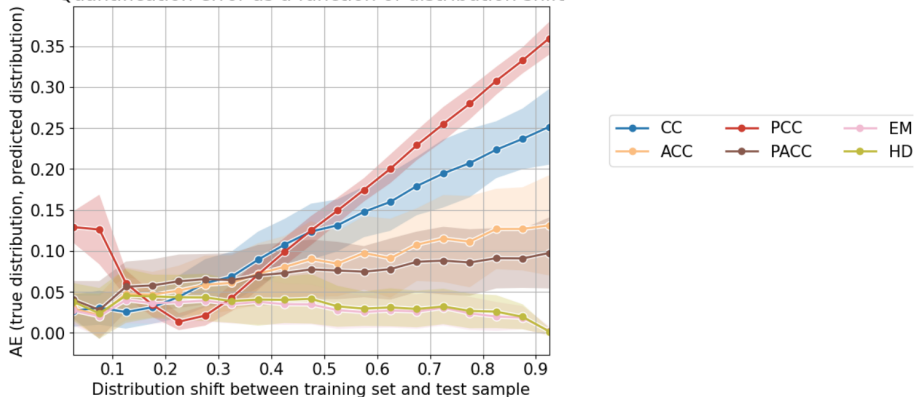
```
qp.plot.binary_bias_global(method_names, true_prevs, estim_prevs)
```



Comparing different models: Plots

```
gp.plot_error_by_drift(method_names, true_prevs, estim_prevs, tr_prevs, show_density=False, show_std=True)
```

Quantification error as a function of distribution shift



Full Example: T1A@LeQua

```

import numpy as np
from sklearn.linear_model import LogisticRegression
import quapy as qp
import quapy.functional as F
from quapy.data.datasets import LEQUA2022_SAMPLE_SIZE, fetch_lequa2022
from quapy.evaluation import evaluation_report
from quapy.method.aggregative import EMQ
from quapy.model_selection import GridSearchQ
import pandas as pd

task = 'T1A' # there are 4 tasks (T1A, T1B, T2A, T2B)
qp.envirn['SAMPLE_SIZE'] = LEQUA2022_SAMPLE_SIZE[task]
qp.envirn['N_JOBS'] = -1

training, val_generator, test_generator = fetch_lequa2022(task=task)

quantifier = EMQ(classifier=LogisticRegression())

# model selection
param_grid = {
    'classifier__C': np.logspace(-3, 3, 7),
}

model_selection = GridSearchQ(quantifier, param_grid, protocol=val_generator, error='mrae', refit=False)
quantifier = model_selection.fit(training)

# evaluation
report = evaluation_report(quantifier, protocol=test_generator, error_metrics=['mae', 'mrae'])

# printing results
pd.set_option('display.expand_frame_repr', False)
report['estim-prev'] = report['estim-prev'].map(F.strprev)
print(report)

print('Averaged values:')
print(report.mean())

```

Full Example: T1A@LeQua

```
import quapy as qp
from quapy.data import LabelledCollection
from quapy.method.base import BaseQuantifier

class PCA_LR(BaseQuantifier):

    def fit(self, data: LabelledCollection):
        pass

    def quantify(self, instances):
        pass
```



Full Example: T1A@LeQua

```
import quapy as qp
import quapy.functional as F
from quapy.data import LabelledCollection
from quapy.method.base import BaseQuantifier
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

class PCA_LR(BaseQuantifier):

    def __init__(self, n_components=50, C=1.):
        self.n_components = n_components
        self.C = C

    def fit(self, data: LabelledCollection):
        self.classes_ = data.classes_
        self.pca = PCA(n_components=self.n_components)
        X = self.pca.fit_transform(data.X)
        self.classifier = LogisticRegression(C=self.C).fit(X, data.y)
        return self

    def quantify(self, X):
        X = self.pca.transform(X)
        y_hat = self.classifier.predict(X)
        return F.prevalence_from_labels(y_hat, classes=self.classes_)
```

Full Example: T1A@LeQua

```

import quapy as qp
import quapy.functional as F
from quapy.data.datasets import LEQUA2022_SAMPLE_SIZE, fetch_lequa2022
from quapy.evaluation import evaluation_report
from quapy.method.aggregative import EMQ
from quapy.model_selection import GridSearchQ
import pandas as pd

from mymethods import PCA_LR

task = 'T1A' # there are 4 tasks (T1A, T1B, T2A, T2B)
qp.environ['SAMPLE_SIZE'] = LEQUA2022_SAMPLE_SIZE[task]
qp.environ['N_JOBS'] = -1

training, val_generator, test_generator = fetch_lequa2022(task=task)

# quantifier = EMQ(classifier=LogisticRegression())
quantifier = PCA_LR()

# model selection
param_grid = {
    'C': [0.1, 1, 10],
    'n_components': [50, 100]
}

model_selection = GridSearchQ(quantifier, param_grid, protocol=val_generator, error='mrae', refit=False, verbose=True)
quantifier = model_selection.fit(training)

# evaluation
report = evaluation_report(quantifier, protocol=test_generator, error_metrics=['mae', 'mrae'], verbose=True)

# printing results
pd.set_option('display.expand_frame_repr', False)
report['estim-prev'] = report['estim-prev'].map(F.strprev)
print(report)

print('Averaged values:')
print(report.mean())

```

Full Example: Output

```
[GridSearchQ:PCA_LR]: hyperparams={'C': 1, 'n_components': 50} got mrae score 2.24224 [took 7.7348s]
[GridSearchQ:PCA_LR]: hyperparams={'C': 0.1, 'n_components': 50} got mrae score 2.26135 [took 7.8331s]
[GridSearchQ:PCA_LR]: hyperparams={'C': 10, 'n_components': 50} got mrae score 2.26711 [took 7.8997s]
[GridSearchQ:PCA_LR]: hyperparams={'C': 1, 'n_components': 100} got mrae score 1.97783 [took 7.9226s]
[GridSearchQ:PCA_LR]: hyperparams={'C': 10, 'n_components': 100} got mrae score 1.97374 [took 7.9275s]
[GridSearchQ:PCA_LR]: hyperparams={'C': 0.1, 'n_components': 100} got mrae score 1.99763 [took 7.9820s]
[GridSearchQ:PCA_LR]: optimization finished: best params {'C': 10, 'n_components': 100} (score=1.97374)
[took 9.4701s]
```

predicting: 5000it [01:00, 82.73it/s]

	true-prev	estim-prev	mae	mrae
0	[0.308, 0.692]	[0.216, 0.784]	0.092	0.214670
1	[0.896, 0.104]	[0.576, 0.424]	0.320	1.687608
2	[0.848, 0.152]	[0.456, 0.544]	0.392	1.503316
3	[0.016, 0.984]	[0.088, 0.912]	0.072	2.036511
4	[0.728, 0.272]	[0.488, 0.512]	0.240	0.602340
...
4995	[0.720, 0.280]	[0.416, 0.584]	0.304	0.749533
4996	[0.868, 0.132]	[0.508, 0.492]	0.360	1.550180
4997	[0.292, 0.708]	[0.216, 0.784]	0.076	0.182773
4998	[0.240, 0.760]	[0.184, 0.816]	0.056	0.152448
4999	[0.948, 0.052]	[0.604, 0.396]	0.344	3.366238

[5000 rows x 4 columns]

Averaged values:

mae 0.189446

mrae 1.686049

dtype: float64



Wrapping up

- QUAPY is a Python-based open-source tool that supports research and development in quantification
- Automates typical tasks of the quantification workflow
- A rich suite of methods, datasets, and protocols available
- Fully documented

Alejandro Moreo

alejandro.moreo@isti.cnr.it



<https://github.com/HLT-ISTI/QuaPy>