crepes: a Python Package for Conformal Regressors and Predictive Systems

Henrik Boström

bostromh@kth.se

Division of Software and Computer Systems
Department of Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

August 24, 2022

Conformal regressors

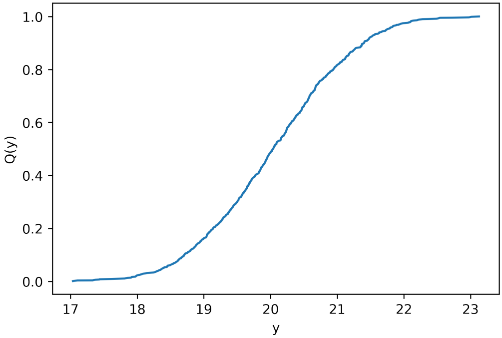
Conformal regressors turn point predictions, output by *any underlying regression model*, into prediction intervals

E.g., the point prediction $\hat{y} = 23.5$ may become $\hat{Y} = [21.0, 25.0]$

Given a confidence level $1 - \epsilon$, the conformal prediction framework guarantees, *with no stronger assumptions than the standard IID*, that the probability of making an error, i.e., the correct target value is not included in the prediction interval, is not larger than ϵ.

Conformal predictive systems

Conformal predictive systems for regression output *conformal predictive distributions* (cumulative distribution functions):



Software packages for conformal regressors and predictive systems

Software package	Lang	Ind	Trans	Agg	OOB	Mond	CPS
nonconformist	Python	✓	✗	✓	✓	✗	✗
Orange3-Conformal	Python	✓	✗	✓	✗	✗	✗
MAPIE	Python	✗	✗	✓	✓	✗	✗
conformalInference	R	✓	✓	✗	✗	✗	✗
crepes	Python	✓	✗	✗	✓	✓	✓

`crepes` (**c**onformal **re**gressors and **p**redictive **s**ystems)

- For documentation, examples, etc., visit: https://github.com/henrikbostrom/crepes
- Installation:
  ```
  pip install crepes
  ```

How to generate prediction intervals/conformal predictive distributions with `crepes`

Input:

- residuals for a calibration set
- point predictions for a test set
- difficulty estimates for the calibration and test set (optional)
- Mondrian categories for the calibration and test set (optional)

Output:

- a prediction interval/conformal predictive distribution for each test object, based on its point prediction, difficulty estimate and Mondrian category

Importing classes and functions

```
from crepes import ConformalRegressor, ConformalPredictiveSystem

from crepes.fillings import (sigma_variance,
                             sigma_variance_oob,
                             sigma_knn,
                             binning)
```

Some useful additional packages and functions

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml
```

Import a dataset

```
dataset = fetch_openml(name="house_sales",version=3)

X = dataset.data.values.astype(float)
y = dataset.target.values.astype(float)

y = np.array([(y[i]-y.min())/(y.max()-y.min()) for i in range(len(y))])

display(X.shape)
```

```
(21613, 21)
```

Divide the data, train a model and get residuals

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

X_prop_train, X_cal, y_prop_train, y_cal = train_test_split(X_train, y_train,
                                                            test_size=0.25)

learner_prop = RandomForestRegressor(n_jobs=-1, n_estimators=500)

learner_prop.fit(X_prop_train, y_prop_train)

y_hat_cal = learner_prop.predict(X_cal)

residuals_cal = y_cal - y_hat_cal
```

Standard conformal regressors

```
cr_std = ConformalRegressor()

cr_std.fit(residuals=residuals_cal)

y_hat_test = learner_prop.predict(X_test)

intervals = cr_std.predict(y_hat=y_hat_test, confidence=0.99)

display(intervals)
```

```
array([[-0.03201615,  0.11384434],
       [-0.04521853,  0.10064197],
       [-0.00424899,  0.14161151],
       ...,
       [ 0.02452438,  0.17038488],
       [-0.04886076,  0.09699974],
       [-0.03928885,  0.10657165]])
```

Excluding impossible values

```python
intervals_std = cr_std.predict(y_hat=y_hat_test, y_min=0, y_max=1)

display(intervals_std)
```

```
array([[0.01106069, 0.0707675 ],
       [0.        , 0.05756512],
       [0.03882786, 0.09853466],
       ...,
       [0.06760123, 0.12730803],
       [0.        , 0.05392289],
       [0.003788  , 0.0634948 ]])
```

Normalized conformal regressors using kNN

```
sigmas_cal_knn = sigma_knn(X=X_cal, residuals=residuals_cal)

cr_norm_knn = ConformalRegressor()

cr_norm_knn.fit(residuals=residuals_cal, sigmas=sigmas_cal_knn)

sigmas_test_knn = sigma_knn(X=X_cal, residuals=residuals_cal, X_test=X_test)

intervals_norm_knn = cr_norm_knn.predict(y_hat=y_hat_test,
                                         sigmas=sigmas_test_knn,
                                         y_min=0, y_max=1)

display(intervals_norm_knn)
```

```
array([[0.02213197, 0.05969621],
       [0.00490446, 0.05051898],
       [0.03095047, 0.10641205],
       ...,
       [0.04670171, 0.14820755],
       [0.0012564 , 0.04688257],
       [0.0084359 , 0.0588469 ]])
```

Normalized conformal regressors using variance

```python
sigmas_cal_var = sigma_variance(X=X_cal, learner=learner_prop)

cr_norm_var = ConformalRegressor()

cr_norm_var.fit(residuals=residuals_cal, sigmas=sigmas_cal_var)

sigmas_test_var = sigma_variance(X=X_test, learner=learner_prop)

intervals_norm_var = cr_norm_var.predict(y_hat=y_hat_test,
                                         sigmas=sigmas_test_var,
                                         y_min=0, y_max=1)

display(intervals_norm_var)
```

```
array([[0.01393936, 0.06788882],
       [0.00069462, 0.05472882],
       [0.04112209, 0.09624044],
       ...,
       [0.07009253, 0.12481673],
       [0.        , 0.05101747],
       [0.00658782, 0.06069498]])
```

Mondrian conformal regressors

```python
bins_cal, bin_thresholds = binning(values=sigmas_cal_knn, bins=20)

cr_mond = ConformalRegressor()

cr_mond.fit(residuals=residuals_cal, bins=bins_cal)

bins_test = binning(values=sigmas_test_knn, bins=bin_thresholds)

intervals_mond = cr_mond.predict(y_hat=y_hat_test, bins=bins_test,
                                 y_min=0, y_max=1)

display(intervals_mond)
```

```
array([[0.02092019, 0.06090799],
       [0.01116473, 0.04425871],
       [0.        , 0.14988479],
       ...,
       [0.        , 0.19704537],
       [0.0075225 , 0.04061647],
       [0.01270447, 0.05457833]])
```

Out-of-bag (OOB) residuals

```
learner_full = RandomForestRegressor(n_jobs=-1, n_estimators=500,
                                     oob_score=True)

learner_full.fit(X_train, y_train)

oob_predictions = learner_full.oob_prediction_

residuals_oob = y_train - oob_predictions
```

Standard conformal regressors using OOB residuals

```
cr_std_oob = ConformalRegressor()

cr_std_oob.fit(residuals=residuals_oob)

y_hat_full = learner_full.predict(X_test)

intervals_std_oob = cr_std_oob.predict(y_hat=y_hat_full, y_min=0, y_max=1)

display(intervals_std_oob)
```

```
array([[0.01086903, 0.0704304 ],
       [0.        , 0.058473  ],
       [0.03861869, 0.09818006],
       ...,
       [0.06603016, 0.12559153],
       [0.        , 0.05423123],
       [0.00529826, 0.06485964]])
```

Normalized conformal regressors using kNN and OOB residuals

```
sigmas_oob_knn = sigma_knn(X=X_train, residuals=residuals_oob)

cr_norm_knn_oob = ConformalRegressor()

cr_norm_knn_oob.fit(residuals=residuals_oob, sigmas=sigmas_oob_knn)

sigmas_test_knn_oob = sigma_knn(X=X_train, residuals=residuals_oob,
                                X_test=X_test)

intervals_norm_knn_oob = cr_norm_knn_oob.predict(y_hat=y_hat_full,
                                                 sigmas=sigmas_test_knn_oob,
                                                 y_min=0, y_max=1)

display(intervals_norm_knn_oob)
```

```
array([[0.0190055 , 0.06229394],
       [0.00482501, 0.05255961],
       [0.03208473, 0.10471402],
       ...,
       [0.073753  , 0.1178687 ],
       [0.00489781, 0.04400328],
       [0.01673711, 0.05342079]])
```

Normalized conformal regressors using variance and OOB residuals

```
sigmas_oob_var = sigma_variance_oob(X=X_train, learner=learner_full)

cr_norm_var_oob = ConformalRegressor()

cr_norm_var_oob.fit(residuals=residuals_oob, sigmas=sigmas_oob_var)

sigmas_test_var_oob = sigma_variance_oob(X=X_test,
                                          learner=learner_full)

intervals_norm_var_oob = cr_norm_var_oob.predict(y_hat=y_hat_full,
                                                  sigmas=sigmas_test_var_oob,
                                                  y_min=0, y_max=1)

display(intervals_norm_var_oob)
```

```
array([[0.01395335, 0.06734608],
       [0.00197646, 0.05540815],
       [0.04103998, 0.09575876],
       ...,
       [0.06871305, 0.12290864],
       [0.        , 0.05111519],
       [0.00831553, 0.06184237]])
```

Mondrian conformal regressors using OOB residuals

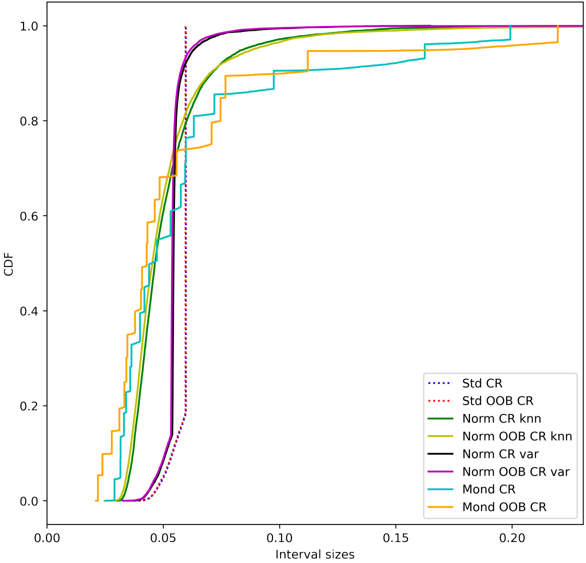
```
bins_oob, bin_thresholds_oob = binning(values=sigmas_oob_knn, bins=20)

cr_mond_oob = ConformalRegressor()

cr_mond_oob.fit(residuals_oob, bins=bins_oob)

bins_test_oob = binning(values=sigmas_test_knn_oob, bins=bin_thresholds_oob)

intervals_mond_oob = cr_mond_oob.predict(y_hat=y_hat_full,
                                         bins=bins_test_oob,
                                         y_min=0, y_max=1)

display(intervals_mond_oob)
```

```
array([[0.02172422, 0.05957522],
       [0.00710548, 0.05027914],
       [0.01234326, 0.12445548],
       ...,
       [0.07561287, 0.11600883],
       [0.00889176, 0.04000933],
       [0.02114419, 0.04901371]])
```

Coverage and interval sizes

	Coverage	Mean size	Median size
Std CR	0.9473	0.0585	0.0597
Std OOB CR	0.9469	0.0583	0.0596
Norm CR knn	0.9482	0.0519	0.0464
Norm OOB CR knn	0.9477	0.0513	0.0453
Norm CR var	0.9478	0.0552	0.0544
Norm OOB CR var	0.9456	0.0546	0.0538
Mond CR	0.9527	0.0596	0.0450
Mond OOB CR	0.9509	0.0560	0.0429
Mean	0.9484	0.0557	0.0509

Distribution of interval sizes



Evaluating a conformal regressor

```
cr_mond_oob.evaluate(y_hat=y_hat_full, y=y_test, sigmas=sigmas_test_knn_oob,
                bins=bins_test_oob, confidence=0.99, y_min=0, y_max=1)
```

```
{'error': 0.008513000832793605,
 'efficiency': 0.09592156405107215,
 'time_fit': 0.0008156299591064453,
 'time_evaluate': 0.0008301734924316406}
```

Evaluation

	Std OOB CR		Norm OOB CR knn		Mond OOB CR	
	0.95	**0.99**	**0.95**	**0.99**	**0.95**	**0.99**
error	0.0531	0.0095	0.0523	0.0097	0.0491	0.0085
efficiency	0.0583	0.1191	0.0513	0.0886	0.0560	0.0959
time_fit	0.0008	0.0008	0.0005	0.0005	0.0008	0.0008
time_evaluate	0.0003	0.0001	0.0001	0.0001	0.0006	0.0005

Standard and normalized conformal predictive systems

```
cps_std = ConformalPredictiveSystem().fit(residuals=residuals_cal)

cps_norm = ConformalPredictiveSystem().fit(residuals=residuals_cal,
                                           sigmas=sigmas_cal_knn)

cps_std_oob = ConformalPredictiveSystem().fit(residuals=residuals_oob)

cps_norm_oob = ConformalPredictiveSystem().fit(residuals=residuals_oob,
                                               sigmas=sigmas_oob_knn)
```

Mondrian conformal predictive systems

```
bins_cal, bin_thresholds = binning(values=y_hat_cal, bins=5)

cps_mond_std = ConformalPredictiveSystem().fit(residuals=residuals_cal,
                                               bins=bins_cal)

cps_mond_norm = ConformalPredictiveSystem().fit(residuals=residuals_cal,
                                                sigmas=sigmas_cal_knn,
                                                bins=bins_cal)

bins_test = binning(values=y_hat_test, bins=bin_thresholds)
```

Mondrian conformal predictive systems using OOB residuals

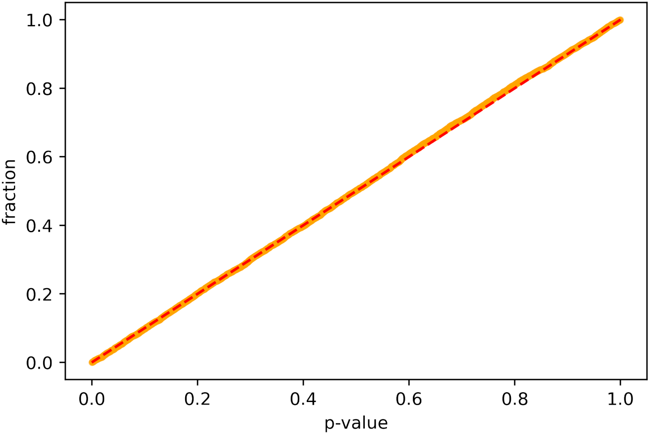
```
bins_oob, bin_thresholds_oob = binning(values=oob_predictions, bins=5)

cps_mond_std_oob = ConformalPredictiveSystem().fit(residuals=residuals_oob,
                                                   bins=bins_oob)

cps_mond_norm_oob = ConformalPredictiveSystem().fit(residuals=residuals_oob,
                                                    sigmas=sigmas_oob_knn,
                                                    bins=bins_oob)

bins_test_oob = binning(values=y_hat_full, bins=bin_thresholds_oob)
```

Obtaining p-values

```
p_values = cps_mond_norm_oob.predict(y_hat=y_hat_full,
                                     sigmas=sigmas_test_knn_oob,
                                     bins=bins_test_oob,
                                     y=y_test)

display(p_values)
```

```
array([[0.98879981],
       [0.81539093],
       [0.32837239],
       ...,
       [0.07967211],
       [0.1247485 ],
       [0.13419304]])
```

Distribution of p-values



Obtaining threshold values

```
thresholds = cps_mond_norm_oob.predict(y_hat=y_hat_full,
                                       sigmas=sigmas_test_knn_oob,
                                       bins=bins_test_oob,
                                       higher_percentiles=99)

display(thresholds)
```

```
array([[0.06055307],
       [0.04766426],
       [0.11890169],
       ...,
       [0.15595684],
       [0.03999288],
       [0.05194553]])
```

Obtaining p-values and thresholds

```
results = cps_mond_norm_oob.predict(y_hat=y_hat_full,
                                    sigmas=sigmas_test_knn_oob,
                                    bins=bins_test_oob,
                                    y=y_test,
                                    lower_percentiles=[2.5, 5],
                                    higher_percentiles=[95, 97.5])

display(results)
```

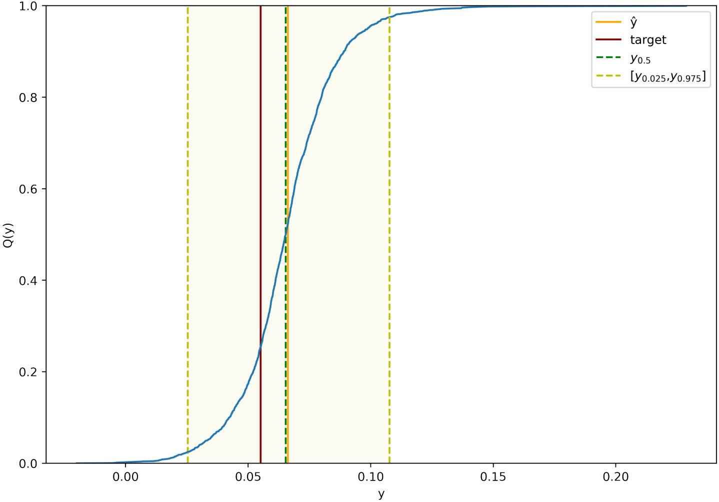
```
array([[0.98888734, 0.02654675, 0.0286026 , 0.05104849, 0.05516526],
       [0.81512743, 0.01376471, 0.01657479, 0.03859596, 0.04181158],
       [0.32805392, 0.03245515, 0.0391414 , 0.09691522, 0.10471402],
       ...,
       [0.07992451, 0.06116837, 0.06930916, 0.12472251, 0.13620793],
       [0.12469454, 0.01222145, 0.01452355, 0.03256389, 0.03519821],
       [0.13453065, 0.02312775, 0.02486993, 0.04389112, 0.04737977]])
```

Obtaining conformal predictive distributions

```
cpds = cps_mond_norm.predict(y_hat=y_hat_test,
                             sigmas=sigmas_test_knn,
                             bins=bins_test,
                             return_cpds=True)

display(len(y_hat_test))
display(cpds.shape)
```

```
10807
(10807,)
```

Conformal predictive distribution



Evaluating conformal predictive systems

```python
cps_mond_norm_oob.evaluate(y_hat=y_hat_full,
                           bins=bins_test_oob,
                           sigmas=sigmas_test_knn_oob,
                           y=y_test,
                           confidence=0.95)
```

```
{'error': 0.04524845007865275,
 'efficiency': 0.05382280610818563,
 'CRPS': 0.006837617321473148,
 'time_fit': 0.0009062290191650391,
 'time_evaluate': 0.4419891834259033}
```

Evaluation

	Std OOB CPS		Norm OOB CPS		Mond norm OOB CPS	
	0.95	**0.99**	**0.95**	**0.99**	**0.95**	**0.99**
error	0.0538	0.0100	0.0512	0.0082	0.0452	0.0089
efficiency	0.0598	0.1253	0.0518	0.0944	0.0537	0.0889
CRPS	0.0073	0.0073	0.0070	0.0070	0.0068	0.0068
time_fit	0.0004	0.0004	0.0004	0.0004	0.0008	0.0008
time_evaluate	0.9449	0.7943	0.8323	0.8320	0.3751	0.3847

Concluding remarks

- The Python package `crepes` has been presented, which allows for generating, applying and evaluating conformal regressors and predictive systems
- The computational cost can be kept very low thanks to the use of the NumPy library; the only significant cost involves evaluations with respect to continuous ranked probability score (CRPS)
- Residuals, difficulty estimates and Mondrian categories are assumed to be calculated separately from the package; some standard options have been included in `crepes.fillings`
- For more information, consult
  https://github.com/henrikbostrom/crepes
  Contributions/suggestions are most welcome!