

HydrEns_eval Manual

USER'S GUIDE

Contents

HydrEns_eval Structure.....	1
Engine	1
Data generation	3
Evaluation.....	4
Rainfall Evaluation Example	7
Low level data access examples	11
Data creation	11
Data handling and representation	12
Runoff Evaluation Example	14
Forecast data preprocessing.....	14
Observation Data Collection	16
Contingency table-based analysis.....	16
Full ensemble analysis.....	18

List of Figures

Figure 1: UML Class diagram for HydrEns_eval engine	1
Figure 2: Event file structure	3
Figure 3: data_collection execution	3
Figure 4: create lead time arrays execution	4
Figure 5: load_catchment function	5
Figure 6: paths file structure	5
Figure 7: Evaluation workflow	6
Figure 8: quantile area under ROC calculation	7
Figure 9: eval_ROC submodule results	8
Figure 10: eval_ROC execution	8
Figure 11: eval_FullEns submodule results (rainfall)	9
Figure 12: eval_FullEns submodule results (runoff)	9
Figure 13: eval_FullEns execution	10
Figure 14: eval_Cont submodule results	10
Figure 15: fr_to_netcdf_tools example	11
Figure 16: Observation class example	12
Figure 17: Ensemble run class example	13
Figure 18: Runoff data directory example	14
Figure 19: DeHM output example	15
Figure 20: runoff_datagen execution example	15
Figure 21: runoff_analysis execution example	16
Figure 22: Contingency table Hits Adorf	16
Figure 23: Contingency table Misses Adorf	17
Figure 24: Contingency table False Alarms Adorf	17
Figure 25: runoff analysis total ensemble execution example	18
Figure 26: runoff_analysis__tot submodule results	18

List of Tables

Table 1: HydrEns_eval dependencies	2
Table 2: Adopted test regions	4

HydrEns_eval Structure

The Evaluation model consists of three modules: engine, data generation and Evaluation

The data generation module is a preprocessing module which is responsible for both data collection from the original forecast files, creation of the forecast by lead time files and storing them in their corresponding directories.

The Evaluation module is where the performance metrics are calculated and the evaluation workflow for any number of catchments is executed.

The two modules are built on the Engine module, where forecast, observation, and evaluation metrics classes are defined.

Engine

The Engine module contains the set of classes defining the radar/ gauge observations as well as the deterministic and ensemble forecast. The module also contains the set of evaluation metrics required to perform the comparison between the forecasts and the ground truths. This set of metrics includes contingency table class and the Relative Operation Characteristic -ROC- curve, in addition to full ensemble evaluation metrics such as normalized RMSE and CRPS.

Figure 1 shows how the engine is built and what classes are dependent on which.

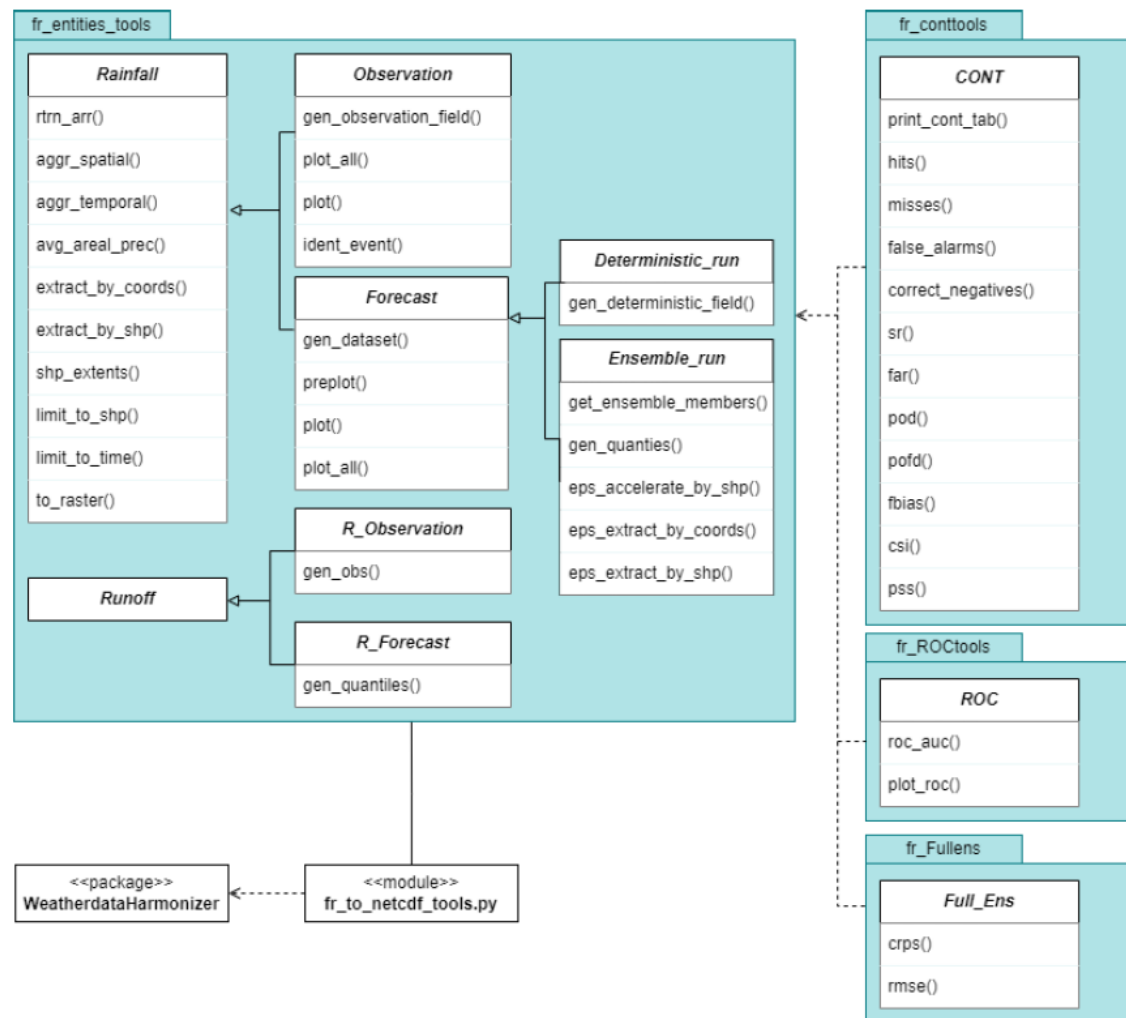


Figure 1: UML Class diagram for HydrEns_eval engine

The responsibilities of each module are as follows:

- `fr_to_netcdf_tools.py`
 - Includes all the functions to convert from the .grib files by the DWD to netCDF format, to be later used as input for the other modules.
- `fr_entities_tools.py`
 - Includes all the forecast and observation objects and methods to be used for evaluation.
- `fr_Conttools.py`
 - Includes the contingency object that utilizes the forecast and observation objects to perform the comparison.
- `fr_ROCtools.py`
 - Includes the relative operation characteristics curve object and plotting method.
- `fr_Fullens.py`
 - Includes the continuous rank probability score and root mean square methods that applies for full ensemble evaluation.

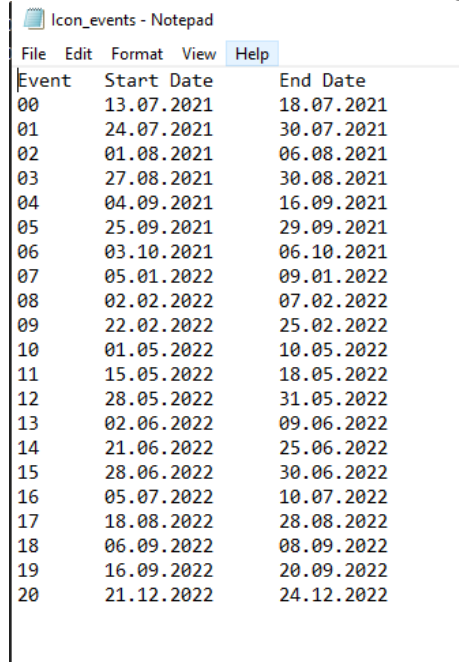
HydrEns_eval engine and other modules are available online for free and can be retrieved at https://github.com/MohamedElGhorab95/HydrEns_eval and has the following dependencies shown in Table 1 that are essential for smooth execution while avoiding versioning problems.

Table 1: HydrEns_eval dependencies

Package	Version
Weather Data Harmonizer	-
Xarray	2023.4.2
Xskillscore	0.00.24
Geopandas	0.12.02
Pyshp	2.03.01
Pyproj	3.05.00
Odc-geo	0.03.03
Rioxarray	0.14.01
Pandas	1.04.00

Data generation

This module consists of two submodules, `data_collection`, and `create_lt_data`. The `data_collection` module requires input `.txt` files for the events needed to be included in the evaluation. The file structure should follow the one depicted in Figure 2.



Event	Start Date	End Date
00	13.07.2021	18.07.2021
01	24.07.2021	30.07.2021
02	01.08.2021	06.08.2021
03	27.08.2021	30.08.2021
04	04.09.2021	16.09.2021
05	25.09.2021	29.09.2021
06	03.10.2021	06.10.2021
07	05.01.2022	09.01.2022
08	02.02.2022	07.02.2022
09	22.02.2022	25.02.2022
10	01.05.2022	10.05.2022
11	15.05.2022	18.05.2022
12	28.05.2022	31.05.2022
13	02.06.2022	09.06.2022
14	21.06.2022	25.06.2022
15	28.06.2022	30.06.2022
16	05.07.2022	10.07.2022
17	18.08.2022	28.08.2022
18	06.09.2022	08.09.2022
19	16.09.2022	20.09.2022
20	21.12.2022	24.12.2022

Figure 2: Event file structure

The module then reads the given events and collects the `.grb` data then converts them to netCDF file format and automatically stores them in the Folder `[HydrEns_eval/Data/]`. This process is executed for the forecast files as well as the radar observations at the same time.

```
#####
if __name__ == '__main__':
#####
# getting all datapoint indices covering the extents of Radolan-RW raster for
# Sachsen after WGS84 > lat 50.1 - 51.8 | long 11.7 - 15.2 <

lon, lat, id_lon, id_lat = fr_to_netcdf_tools.target(50.1,51.8,11.7,15.2,"radolanrx",version=4)
#####

tic = time.time()
create_radar('in/Cosmo_events.txt')
print((time.time()-tic)/3600) # operation time in hours

tic = time.time()
create_forecast('in/Cosmo_events.txt','COSMO',"D")
print((time.time()-tic)/3600) # operation time in hours

tic = time.time()
create_forecast('in/Cosmo_events.txt','COSMO',"eps")
print((time.time()-tic)/3600) # operation time in hours

tic = time.time()
create_radar('in/Icon_events.txt')
print((time.time()-tic)/3600) # operation time in hours

tic = time.time()
create_forecast('in/Icon_events.txt','ICON',"D")
print((time.time()-tic)/3600) # operation time in hours

tic = time.time()
create_forecast('in/Icon_events.txt','ICON',"eps")
print((time.time()-tic)/3600) # operation time in hours
```

Figure 3: data_collection execution

Note: The target grid here is predefined to the extents of Saxony. Which means the resulting netCDF files will be limited to the hardcoded coordinates after WGS84.

The create_lt_data module slices the already created netCDF files and creates more netCDF files based on the forecast lead time, with the goal of having separate arrays for lead times 3,6,9,12,15,18,21,24 hrs to be later used in the forecast quality evaluation. Figure 4 shows how the functions of this module should be executed.

```
#####
if __name__ == '__main__':
#####

    events = load_events('Cosmo_events.txt')

    for lt in np.arange(3,25,3):
        process_lead(lt, 'cosmod2eps_ev.nc', 3, 'cosmod2eps', events)
```

Figure 4: create lead time arrays execution

The resulting lead time arrays are then automatically stored in respective folders in [HydrEns_eval/Data/NetCDFs/]

Evaluation

The Evaluation module contains three submodules each calculating different set of metrics, the area under the ROC, the Contingency tables and several related metrics, and finally metrics assessing the performance of the ensemble as a whole.

These three submodules perform the evaluation exclusively for the catchments shown in Table 2. These catchments are hardcoded in the load_catchment function as shown in Figure 5. This function retrieves the path for the shapefile of the catchment from the '.txt' file [HydrEns_eval/in/paths.txt] to be used in the dataset cropping and limiting the calculation extents.

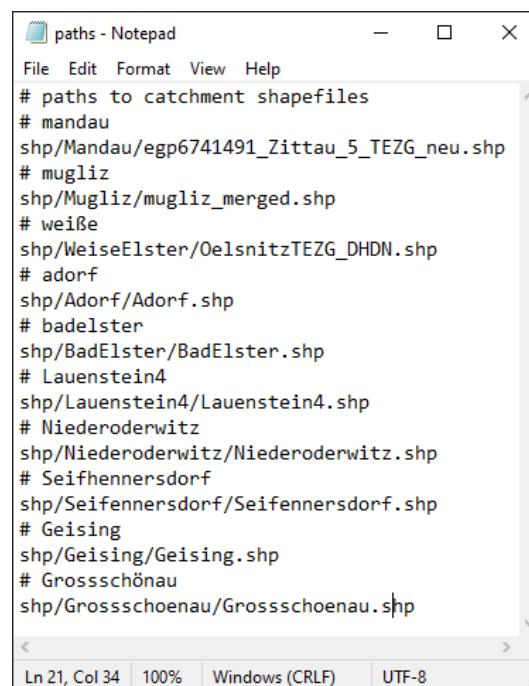
Table 2: Adopted test regions

Region	Catchment
Weiße Elster	Bad Elster
	Adorf
	Oelsnitz
Mandau	Niederoderwitz
	Seifhennersdorf
	Großschönau
	Zittau
Müglitz	Geising
	Lauenstein
	Dohna

```
def load_catchment(name):
    # loading the catchment of interest
    source_file_path = "in/paths.txt"
    with open(source_file_path, 'r') as f:
        # Read all the lines in the file
        lines = f.readlines()
        if name == 'Zittau':
            return lines[2].strip()
        if name == 'Dohna':
            return lines[4].strip()
        if name == 'Oelsnitz':
            return lines[6].strip()
        if name == 'Adorf':
            return lines[8].strip()
        if name == 'Bad Elster':
            return lines[10].strip()
        if name == 'Lauenstein':
            return lines[12].strip()
        if name == 'Niederoderwitz':
            return lines[14].strip()
        if name == 'Seifhennersdorf':
            return lines[16].strip()
        if name == 'Geising':
            return lines[18].strip()
        if name == 'Grossschoenau':
            return lines[20].strip()
```

Figure 5: load_catchment function

Note: If other catchments are to be added to the analysis, they should be added to the loading function as well as the paths file with the format shown in Figure 6. And the actual shape file should be stored in [HydrEns_eval/shp].



```
paths - Notepad
File Edit Format View Help
# paths to catchment shapefiles
# mandau
shp/Mandau/egp6741491_Zittau_5_TEGZ_neu.shp
# mugliz
shp/Mugliz/mugliz_merged.shp
# weiße
shp/WeisseElster/OelsnitzTEZG_DHDN.shp
# adorf
shp/Adorf/Adorf.shp
# badelster
shp/BadElster/BadElster.shp
# Lauenstein4
shp/Lauenstein4/Lauenstein4.shp
# Niederoderwitz
shp/Niederoderwitz/Niederoderwitz.shp
# Seifhennersdorf
shp/Seifhennersdorf/Seifhennersdorf.shp
# Geising
shp/Geising/Geising.shp
# Grossschoenau
shp/Grossschoenau/Grossschoenau.shp
Ln 21, Col 34 100% Windows (CRLF) UTF-8
```

Figure 6: paths file structure

In addition to the `load_catchment` function the three submodules share the same framework for the evaluation. Figure 7 shows the adopted workflow and the next section discusses in detail how each submodule should be used to produce results.

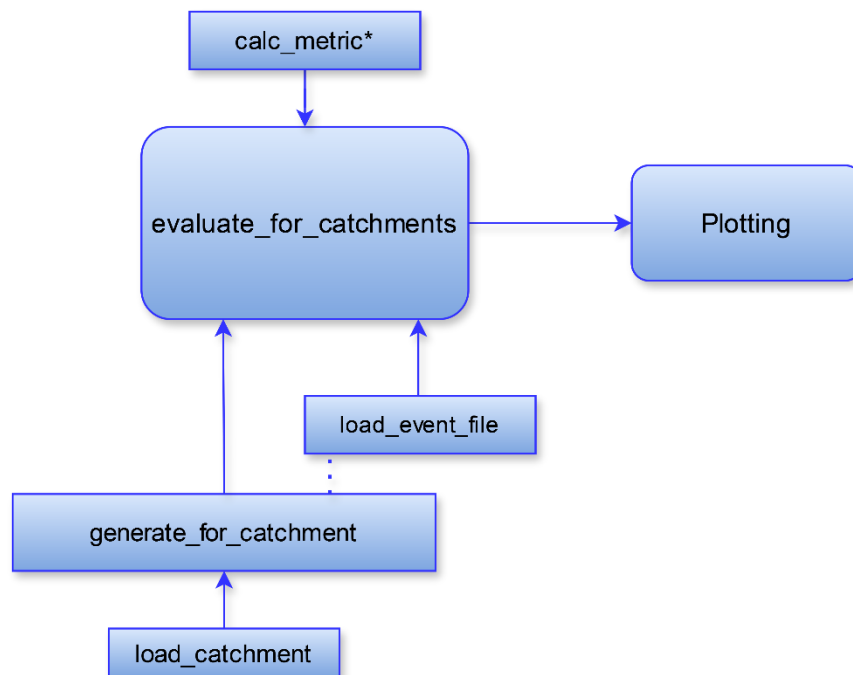


Figure 7: Evaluation workflow

*Note: The `calc_metric` function is marked with an * because it exists only in the contingency table evaluation submodule, while for the other submodules, the metrics are calculated inside the `evaluate_for_catchment` function.*

Rainfall Evaluation Example

The first evaluation submodule, [\[HydrEns_eval/Rainfall_Evaluation/eval_ROC.py\]](#), calculates the area under the ROC curve for each one of the 20 ensemble members for lead times of 3,6,9,12,15,18,21,24 hours. It then calculates the quantiles of the areas for each lead time for both COSMOD2 EPS and ICOND2 EPS as shown in the snippet depicted in Figure 8.

```
def evaluate_for_catchment(cats):
    # TODO change
    max_lead = 24

    variables = [ 'ICOND2EPS q10%', 'ICOND2EPS q25%', 'ICOND2EPS q50%', 'ICOND2EPS q75%', 'ICOND2EPS q90%',
                  'COSMOD2EPS q10%', 'COSMOD2EPS q25%', 'COSMOD2EPS q50%', 'COSMOD2EPS q75%', 'COSMOD2EPS q90%', ]

    roc_auc_dic = {key: [None]*int(max_lead/3) for key in variables}

    c = 0 # leadtime index counter

    for lead in range(3,max_lead+1,3):
        print('Catchment: {} \nCalculating for Lead time: {}hrs'.format(cats,lead),flush= True)

        # loading the files
        files = load_event_file(lead)
        # creating the instances for a specific catchment
        lib_ic, lib_cos = gen_for_catchment(cats,files)

        # Create a progress bar
        progress_bar = tqdm(total=10, desc='Calculating evaluation metrics....')

        icon = ROC(lib_ic['radar'], lib_ic['ICOND2EPS'])
        cosm = ROC(lib_cos['radar'], lib_cos['COSMOD2EPS'])

        roc_auc_dic[variables[0]][c] = icon.roc_auc(10)
        progress_bar.update(1)
        roc_auc_dic[variables[1]][c] = icon.roc_auc(25)
        progress_bar.update(1)
        roc_auc_dic[variables[2]][c] = icon.roc_auc(50)
        progress_bar.update(1)
        roc_auc_dic[variables[3]][c] = icon.roc_auc(75)
        progress_bar.update(1)
        roc_auc_dic[variables[4]][c] = icon.roc_auc(90)
        progress_bar.update(1)
        roc_auc_dic[variables[5]][c] = cosm.roc_auc(10)
        progress_bar.update(1)
        roc_auc_dic[variables[6]][c] = cosm.roc_auc(25)
        progress_bar.update(1)
        roc_auc_dic[variables[7]][c] = cosm.roc_auc(50)
        progress_bar.update(1)
        roc_auc_dic[variables[8]][c] = cosm.roc_auc(75)
        progress_bar.update(1)
        roc_auc_dic[variables[9]][c] = cosm.roc_auc(90)
        progress_bar.update(1)

        # Close the progress bar
        progress_bar.close()

        del lib_ic, lib_cos

        c+=1

    return {'Area under ROC curve':roc_auc_dic}
```

Figure 8: quantile area under ROC calculation

The submodule then plots the results showing the development of the forecast performance against the lead time as shown in Figure 9.

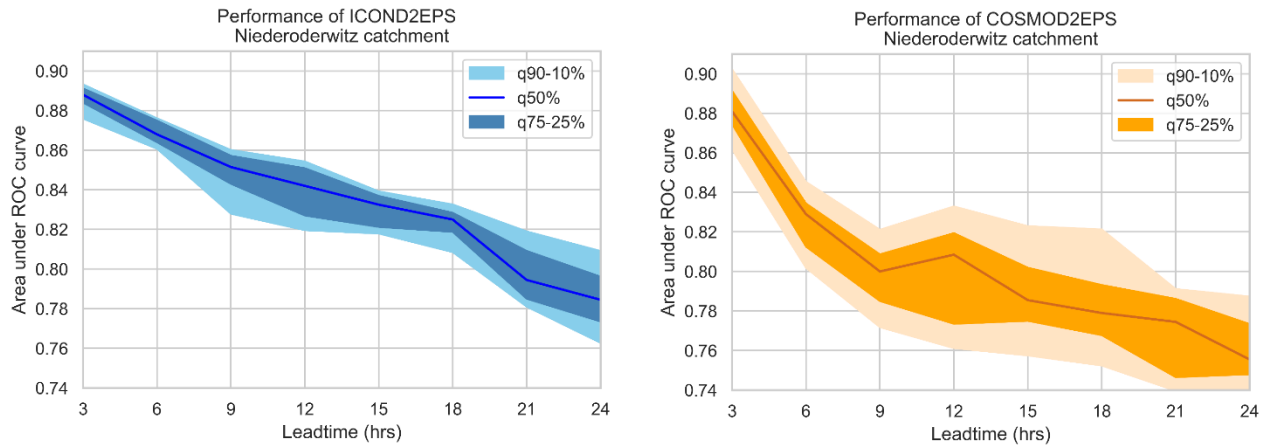


Figure 9: eval_ROC submodule results

The following snippet depicted in Figure 10 shows how this submodule is executed. It is worth mentioning that the catchment list can be modified based on the analysis requirements without the need to modify anything in the module itself.

```
#####
if __name__ == '__main__':
#####

tic = time.time()

cats = ["Dohna", "Oelsnitz", 'Zittau', 'Adorf', 'Geising', 'Grossschoenau',
        'Bad Elster', 'Lauenstein', 'Niederoderwitz', 'Seifhennersdorf']

for cat in cats:
    evaluate_single(cat)

print((time.time()-tic)/3600) # operation time in hours
```

Figure 10: eval_ROC execution

The second evaluation submodule, [\[HydrEns_eval/Rainfall_Evaluation/eval_FullEns.py\]](#), calculates the metrics that reflects the performance of the ensemble as a whole and plots the performance of each region in a separate figure, while showing how the different sized catchments of the same region differ from one another. Figure 11 and Figure 12 show the outcome of this submodule.

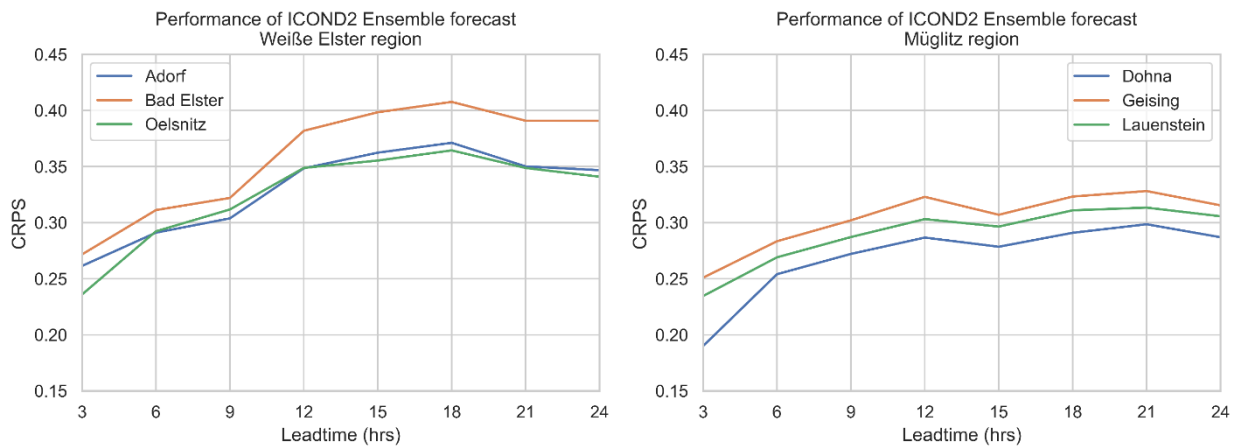


Figure 11: eval_FullEns submodule results (rainfall)

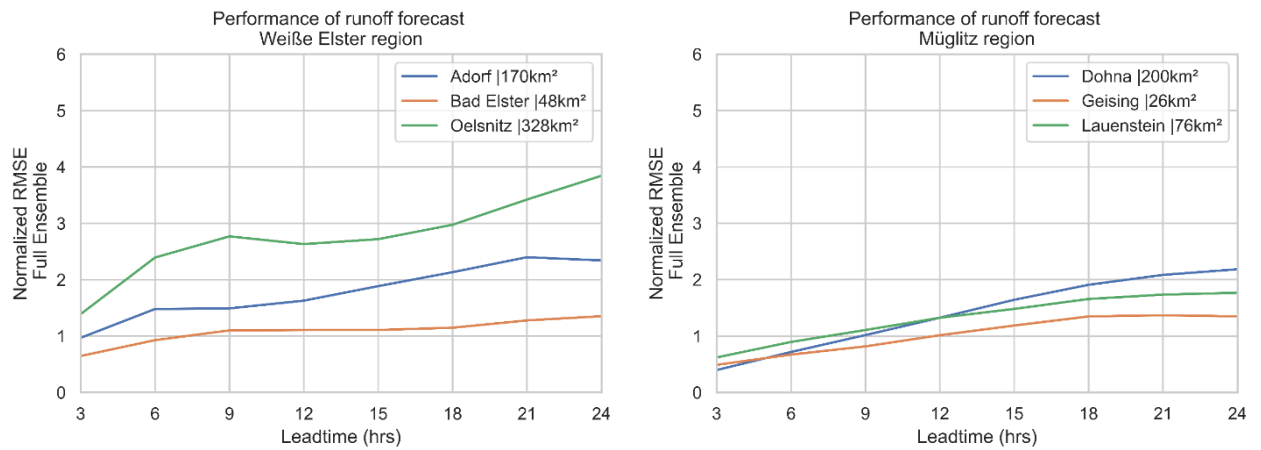


Figure 12: eval_FullEns submodule results (runoff)

This submodule automatically creates the plots for all catchments in all regions. It is executed as shown in Figure 13. In case either a metric or a region is not needed for the analysis it must be manually commented out in the execution section as well as in the script itself.

```
#####
if __name__ == '__main__':
#####

def create_for_regions():

    region1 = {"Oelsnitz":328 , 'Adorf':170 , 'Bad Elster':48}
    region1 = dict(sorted(region1.items()))

    region2 = {'Dohna':200, 'Lauenstein':76, 'Geising':26 }
    region2 = dict(sorted(region2.items()))

    region3 = { 'Zittau':279, 'Grossschoenau': 162, 'Seifhennersdorf':75, 'Niederoderwitz':29 }
    region3 = dict(sorted(region3.items()))

    data1 = {key: [None,region1[key]] for key in region1.keys()}
    for a in data1.keys():
        data1[a][0]=evaluate_for_catchment(a)

    data2 = {key: [None,region2[key]] for key in region2.keys()}
    for a in data2.keys():
        data2[a][0]=evaluate_for_catchment(a)

    data3 = {key: [None,region3[key]] for key in region3.keys()}
    for a in data3.keys():
        data3[a][0]=evaluate_for_catchment(a)

    return [data1, data2, data3]

df = create_for_regions()
plot_curves(df)
```

Figure 13: eval_FullEns execution

The third and last evaluation submodule, [\[HydrEns_eval/Rainfall_Evaluation/eval_Cont.py\]](#), calculates the contingency table and its related metrics, which are the hits, misses, false alarms, accuracy and frequency bias. The submodule then plots the developments of these metrics against the lead time. This is calculated for the main run as well as the 10,25,75,90% quantiles in addition to the mean of the ensemble. Figure 14 shows an example of this submodule output.

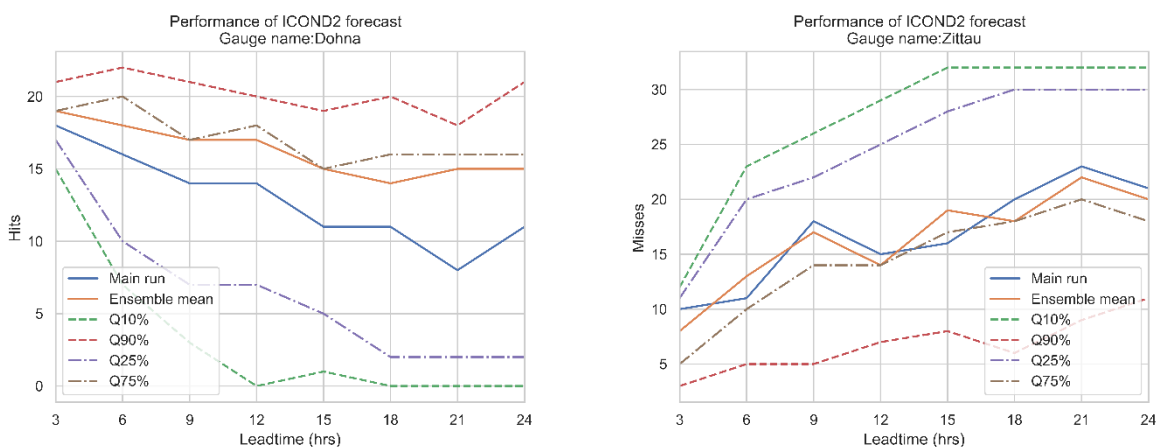


Figure 14: eval_Cont submodule results

Low level data access examples

In this section the low-level use of the classes and methods in the HydrEns_eval engine is demonstrated. The following examples cover the creation and preparation of the forecast files as well as data representation and preparation.

Data creation

The module 'fr_to_netcdf_tools' is used to create netCDF forecast and observation files. The snippet depicted in Figure 15 shows an example for IconD2 / EPS file preparation, however it could be used as well for COSMOD2 / EPS when needed.

```
if __name__ == '__main__':

    # reading the files and exporting

    # getting all datapoint indices covering the extents of Radolan-RW raster for
    # Sachsen after WGS84 > lat 50.1 - 51.8 | long 11.7 - 15.2 <
    lon, lat, id_lon, id_lat = target(50.1,51.8,11.7,15.2,"radolanrx",version=4)

    fortime = timeframe((2021,10,3,0), (2021,10,5,0), "forecast")
    radtime = timeframe((2021,10,3,0), (2021,10,5,0), "radar")

    radolantoNetCDF(radtime, datafolder="//vs-grp08.zih.tu-dresden.de/hwstore/RadolanRW/",
                    idx_lon=id_lon,
                    idx_lat=id_lat,
                    outputfile="Data/NetCDFs/Example/radRW_example.nc")

    IconD2toNetCDF(ST=fortime,datafolder="//vs-grp08.zih.tu-dresden.de/hwstore/IconD2/tot_prec/",
                   longitude=lon,
                   latitude=lat,
                   nearestpoints="Sachsen_nearestpoints.npz",
                   outputfile="Data/NetCDFs/Example/icond2_example.nc")

    IconD2EPStoNetCDF(fortime, "//vs-grp08.zih.tu-dresden.de/hwstore/IconD2eps/tot_prec/",
                       lon,
                       lat,
                       "Sachsen_nearestpoints.npz",
                       "Data/NetCDFs/Example/icond2eps_example.nc")
```

Figure 15: fr_to_netcdf_tools example

Data handling and representation

The module 'fr_entities_tools' reads the netCDF files and stores the data in instances of either Observation or Forecast objects. The module contains also methods for spatial aggregation, plotting, cropping based on shapefiles, limiting data extents, and finally exporting all time steps into raster files.

Figure 16 and Figure 17 show a brief demonstration on what methods could be used for the low-level operations.

```
def test_Obs():
    """
    Testing the Observation Class
    """

    # generate an observation object
    rad = Observation("Data/NetCDFs/Example/radRW_example.nc", 48)
    # generating a rainfall field
    # only needed for manual low level operations, however if the object is to be
    # used in other classes (Cont, ROC) it gets generated automatically
    rad.gen_observation_field()
    # aggregating the rainfall field
    rad_agg = rad.aggr_spatial(3)
    # extracting a part of the observation using geographical coordinates
    sub_rad = Observation("Data/NetCDFs/Example/radRW_example.nc", 48).extract_by_coords(50.02, 51.01, 11.66, 12.68)
    # mean areal rainfall
    rad_avg = rad.avg_areal_prec()

    # Weise Elster
    weise = rad.extract_by_shp("shp/WeiseElster/OelsnitzTEZG_DHDN.shp")

    rad_3 = rad.aggr_temporal(3)

    rad.plot([2021,10,4, 23])
    rad_agg.plot([2021,10,4, 23])
    sub_rad.plot([2021,10,4, 23])

    for i in range(1, 23):
        weise.plot([2021,10,4, i])

    return rad, rad_agg, sub_rad, rad_avg, weise, rad_3
```

Figure 16: Observation class example

```

def test_EPS():
    """
    Testing the Ensemble run Class
    """

    # generate the forecast object
    eps = Ensemble_run("Data/NetCDFs/Example/icond2eps_example.nc", 1)
    # generating a rainfall field for a specific quantile
    # only needed for manual low level operations, however if the object is to be
    # used in other classes (Cont, ROC) it gets generated automatically
    eps95 = eps.gen_quantiles(95) # 95th quantile
    epsavg = eps.gen_quantiles("mean") # mean
    epsmed = eps.gen_quantiles("median") # median

    # extracting a part of the forecast using geographical coordinates
    # the method eps_accelerate_by_shp is used instead of the general extract_by_coords
    # to speed up the calculation process for the smaller areas
    sub95 = eps.eps_accelerate_by_shp("shp/WeiseElster/OelsnitzTEZG_DHDN").gen_quantiles(95)
    subavg = eps.eps_accelerate_by_shp("shp/WeiseElster/OelsnitzTEZG_DHDN").gen_quantiles("mean")
    submed = eps.eps_accelerate_by_shp("shp/WeiseElster/OelsnitzTEZG_DHDN").gen_quantiles("median")

    # mean areal rainfall

    m_eps95 = eps95.avg_areal_prec()
    m_epsavg = epsavg.avg_areal_prec()
    m_epsmed = epsmed.avg_areal_prec().rtrn_arr()

    sub_agg = sub95.aggr_spatial(3).aggr_temporal(3)

    # # the plot method handles plotting ensemble objects,
    # however it is recommended to generate the field first then plot
    eps95.plot([2021,10,4, 23])
    epsavg.plot([2021,10,4, 23])
    epsmed.plot([2021,10,4, 23])
    sub95.plot([2021,10,4, 23])
    subavg.plot([2021,10,4, 23])
    submed.plot([2021,10,4, 23])
    sub_agg.plot([2021,10,4, 23])

    return eps95, epsavg, epsmed, m_eps95, m_epsavg, m_epsmed, sub95, subavg, submed, eps

eps95, epsavg, epsmed, m_eps95, m_epsavg, m_epsmed, sub95, subavg, submed, eps = test_EPS()

```

Figure 17: Ensemble run class example

Note: The 'fr_entities_tools' was modified to be functional in the context of the process mentioned in the Evaluation Framework section. For that reason, some of the methods might not work properly for files created by 'fr_to_netcdf_tools'.

Runoff Evaluation Example

Forecast data preprocessing

First the submodule `[HydrEns_eval/Runoff_Evaluation/runoff_datagen.py]` is used to prepare the netCDF files from the runoff forecast data resulting from the DeHM ensemble simulations. This module should be supplied with a path where all the events to be evaluated are stored. Figure 18 and Figure 19 show an example of how this directory should look like.

Name	Änderungsdatum	Typ	Größe
2021071100	03.08.2023 12:23	Dateiordner	
2021071103	03.08.2023 12:44	Dateiordner	
2021071106	03.08.2023 13:06	Dateiordner	
2021071109	03.08.2023 13:13	Dateiordner	
2021071112	03.08.2023 13:14	Dateiordner	
2021071115	03.08.2023 13:15	Dateiordner	
2021071118	03.08.2023 13:15	Dateiordner	
2021071121	03.08.2023 13:16	Dateiordner	
2021071200	03.08.2023 13:16	Dateiordner	
2021071203	03.08.2023 13:17	Dateiordner	
2021071206	03.08.2023 13:17	Dateiordner	
2021071209	03.08.2023 13:17	Dateiordner	
2021071212	03.08.2023 13:17	Dateiordner	
2021071215	03.08.2023 13:17	Dateiordner	
2021071218	03.08.2023 13:18	Dateiordner	
2021071221	03.08.2023 13:19	Dateiordner	
2021071300	03.08.2023 13:19	Dateiordner	
2021071303	03.08.2023 13:20	Dateiordner	
2021071306	03.08.2023 13:21	Dateiordner	
2021071309	03.08.2023 13:21	Dateiordner	
2021071312	03.08.2023 13:22	Dateiordner	
2021071315	03.08.2023 13:22	Dateiordner	
2021071318	03.08.2023 13:23	Dateiordner	

Figure 18: Runoff data directory example

cast_system > ForData > 2021071103

2021071103 durchsuchen





















Name	Änderungsdatum	Typ	Größe
 2021071103_5371823_Geising1_WeisseMueglitz_data.mat	03.08.2023 12:25	Microsoft Access ...	22 KB
 2021071103_5371823_Geising1_WeisseMueglitz_data.nc	03.08.2023 12:27	NC-Datei	154 KB
 2021071103_5371831_Lauenstein4_Mueglitz_data.mat	03.08.2023 12:25	Microsoft Access ...	23 KB
 2021071103_5371831_Lauenstein4_Mueglitz_data.nc	03.08.2023 12:29	NC-Datei	154 KB
 2021071103_5661311_Adorf_WeisseElster_data.mat	03.08.2023 12:25	Microsoft Access ...	29 KB
 2021071103_5661311_Adorf_WeisseElster_data.nc	03.08.2023 12:35	NC-Datei	154 KB
 2021071103_5661371_Oelsnitz_WeisseElster_data.mat	03.08.2023 12:25	Microsoft Access ...	33 KB
 2021071103_5661371_Oelsnitz_WeisseElster_data.nc	03.08.2023 12:38	NC-Datei	154 KB
 2021071103_53718979_Dohna_Mueglitz_data.mat	03.08.2023 12:25	Microsoft Access ...	23 KB
 2021071103_53718979_Dohna_Mueglitz_data.nc	03.08.2023 12:31	NC-Datei	154 KB
 2021071103_56611313_BadElster_WeisseElster_data.mat	03.08.2023 12:25	Microsoft Access ...	28 KB
 2021071103_56611313_BadElster_WeisseElster_data.nc	03.08.2023 12:33	NC-Datei	154 KB
 2021071103_67414311_Seifhennersdorf_Mandau_data.mat	03.08.2023 12:25	Microsoft Access ...	21 KB
 2021071103_67414311_Seifhennersdorf_Mandau_data.nc	03.08.2023 12:40	NC-Datei	154 KB
 2021071103_67414511_Grossschoenau_Mandau_data.mat	03.08.2023 12:25	Microsoft Access ...	22 KB
 2021071103_67414511_Grossschoenau_Mandau_data.nc	03.08.2023 12:42	NC-Datei	154 KB
 2021071103_67414651_Niederoderwitz_Mandau_data.mat	03.08.2023 12:25	Microsoft Access ...	20 KB
 2021071103_67414651_Niederoderwitz_Mandau_data.nc	03.08.2023 12:44	NC-Datei	154 KB
 2021071103_67414799_Zittau_Mandau_data.mat	03.08.2023 12:25	Microsoft Access ...	22 KB
 2021071103_67414799_Zittau_Mandau_data.nc	03.08.2023 12:46	NC-Datei	154 KB

Figure 19: DeHM output example

As shown in Figure 20 the submodule is executed using the directory where the files are stored as well as the years to be added for the catchments of interest. The submodule then loops through all the folders and concatenates the data accordingly and finally exports the concatenated datasets into netCDF files for each leadtime..

```
# ===== Testing | Examples =====
# =====

if __name__ == '__main__':
    directory = "//vs-grp07.zih.tu-dresden.de/howa/work/students/Mohamed_Elghorab/Mohamed_Elghorab_MSc_Working_files/forecast_system/ForData"

    cat = ['_67414799_Zittau_Mandau_data', '_67414651_Niederoderwitz_Mandau_data',
           '_67414511_Grossschoenau_Mandau_data', '_67414311_Seifhennersdorf_Mandau_data']

    for c in cat:
        gen_runoff_datasets(directory, ['2021', '2022', '2023'], c)

    cat = ['_5661371_Oelsnitz_WeisseElster_data', '_56611313_BadElster_WeisseElster_data',
           '_5661311_Adorf_WeisseElster_data']

    for c in cat:
        gen_runoff_datasets(directory, ['2021', '2022', '2023'], c)

    cat = ['_53718979_Dohna_Mueglitz_data', '_5371831_Lauenstein4_Mueglitz_data',
           '_5371823_Geising1_WeisseMueglitz_data']

    for c in cat:
        gen_runoff_datasets(directory, ['2021', '2022', '2023'], c)
```

Figure 20: runoff_datagen execution example

Observation Data Collection

The observation data is retrieved from <https://www.opensensorweb.de> for the gauging stations of interest. It should be stored as a '.csv' file and named after the gauge and stored in [HydrEns_eval/Runoff_Evaluation]. The Class R_Observation in 'fr_entities_tools' reads these '.csv' files and stores the data as runoff observation object for later evaluation purposes.

Contingency table-based analysis

The submodule [HydrEns_eval/Runoff_Evaluation/runoff_analysis.py] compares the forecasts against the flow observations using predefined thresholds and then calculates and plots the contingency table metrics against the lead times. Figure 21 shows how the submodule is executed while Figure 22 show an example of the output.

```
# =====
##### Testing | Examples #####
# =====

if __name__ == '__main__':
    cat_library = {'Oelsnitz':25.6 , 'Adorf':13.3, 'Bad Elster': 3.72}
    for c in cat_library.keys():
        res = evaluate_for_cat(c,cat_library[c])
        plot_curves(res, c)
```

Figure 21: runoff_analysis execution example

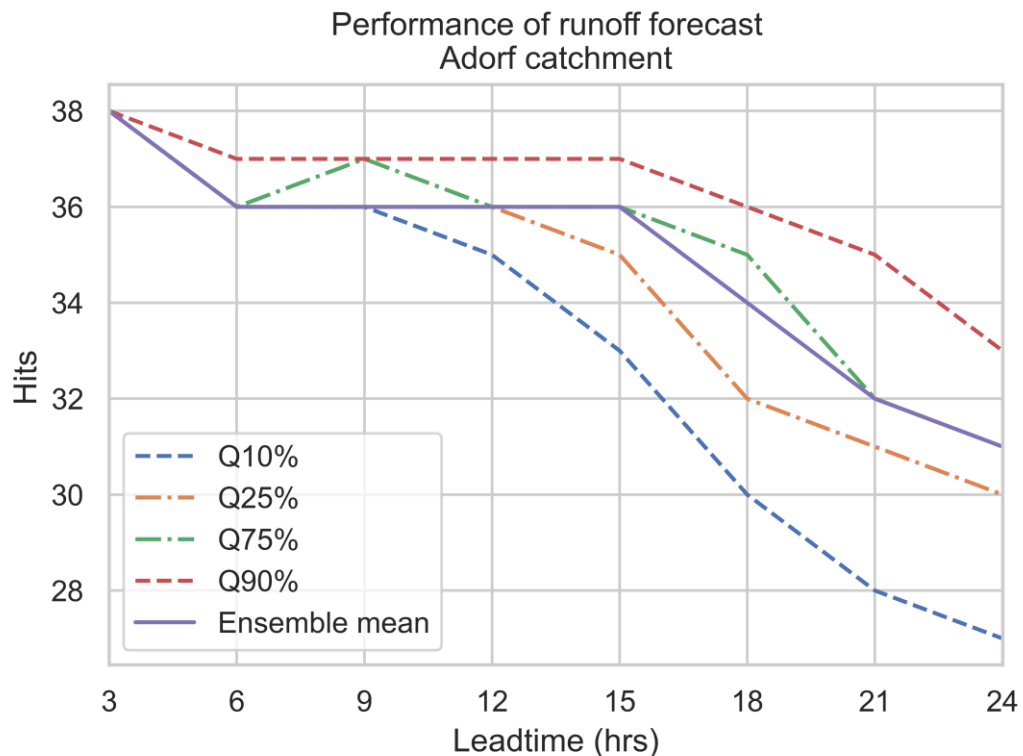


Figure 22: Contingency table Hits | Adorf

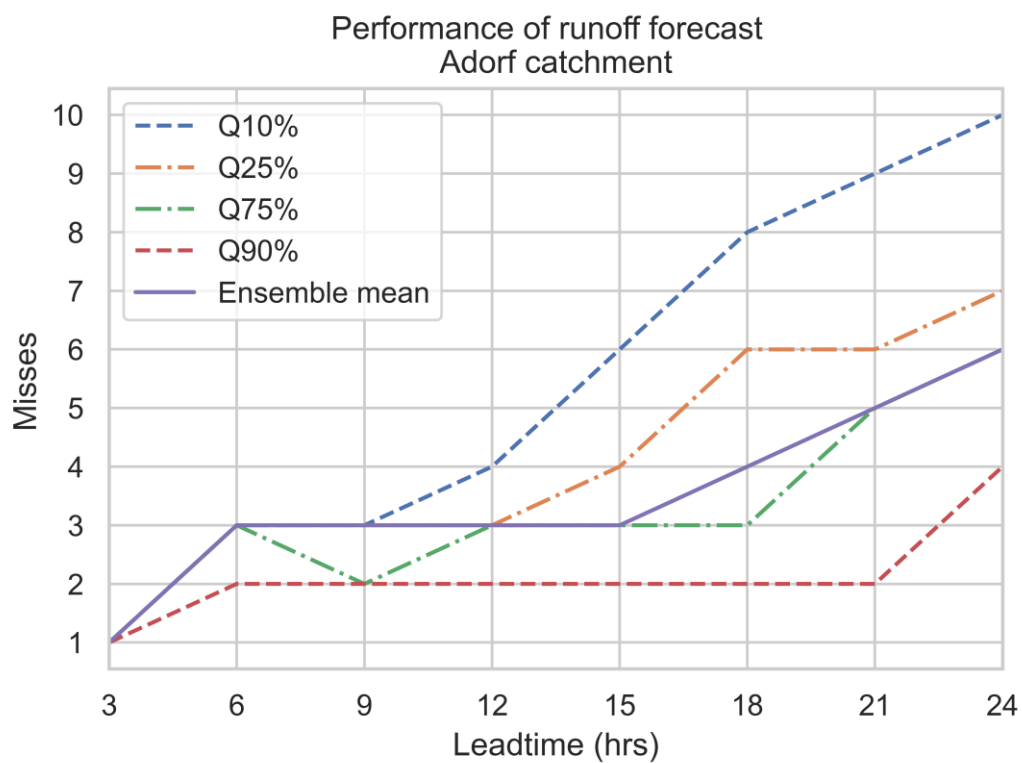


Figure 23:Contingency table Misses | Adorf

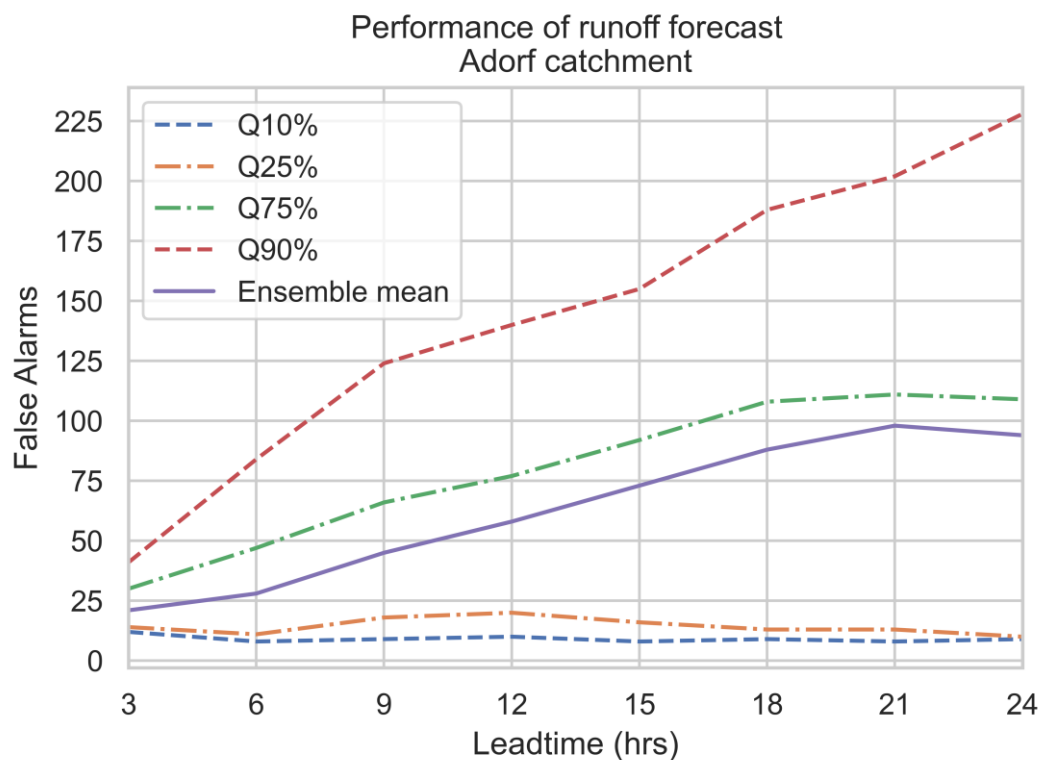


Figure 24:Contingency table False Alarms | Adorf

Note: most of the contents of this submodule were hardcoded. In case of more or different catchments are to be evaluated, they are to be manually added to the script.

Full ensemble analysis

The final submodule [[HydrEns_eval/Runoff_Evaluation/runoff_analysis_tot.py](#)] evaluates the performance of the entire ensemble using CRPS, area under the ROC, and the normalized RMSE and then plots all metrics against the lead time. Figure 25 shows how this submodule can be executed.

```
# =====
##### Testing | Examples #####
# =====

if __name__ == '__main__':

    mets = [ 'CRPS', "Normalized RMSE", 'Area under ROC curve' ]
    # mets = ['Normalized RMSE' ]
    for m in mets:

        df = create_for_regions(m)

        plot_spatial(df,m)
```

Figure 25: runoff analysis total ensemble execution example

This submodule is different from the other ones in the sense that it examines each region on its own and plots all subcatchments of the same region on the same graph, which gives the chance to investigate the relation between the performance of the forecast and the size of the catchment as shown in Figure 26.

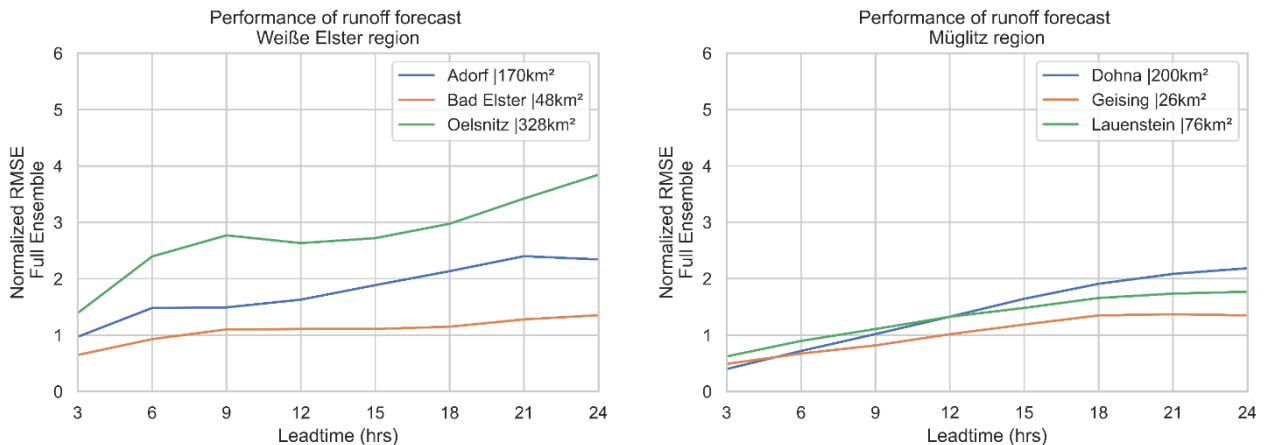


Figure 26: runoff_analysis_tot submodule results