wass2s

Release 0.1.0

Mandela C. M. HOUNGNIBO

May 19, 2025

CONTENTS:

A python-based tool for seasonal climate forecast in West Africa and the Sahel.

The wass2s tool is designed to facilitate implementation of the new generation of seasonal forecasts in West Africa and the Sahel using various statistical and machine learning methods. New generation of seasonal forecasts aligns with the World Meteorological Organization's (WMO) guidelines for objective, operational, and scientifically rigorous seasonal forecasting methods. wass2s helps forecaster to download GCM, reanalysis, and satellite/observation data, build statistical or machine learning models, verify the models using cross-validation, and forecast. A user-friendly jupyter-lab notebook streamlines the forecasting process .

CHAPTER

ONE

FEATURES

- Automated Forecasting
- Reproducibility
- Modularity
- Exploration of Machine Learning Models.

1.1 Installation

1. Create an environment and activate it

- For Windows: download yaml here and run

```
conda env create -f WAS_S2S_windows.yml
conda activate WASS2S
```

- For Linux: download yaml here and run

```
conda env create -f WAS_S2S_linux.yml
conda activate WASS2S
```

2. Install the wass2s package

```
pip install wass2s
```

3. Upgrade the wass2s package

```
pip install --upgrade wass2s
```

1.2 Usage

Comprehensive usage guidelines, including data download, processing, models description, configuration and execution, cross-validation, and verification, are available in the Training Documentation. But for a quick start, use the example notebooks.

Download example notebooks:

```
git clone https://github.com/hmandela/WASS2S_notebooks.git
```

or download the zip file:

```
wget https://github.com/hmandela/WASS2S_notebooks/archive/refs/heads/main.zip -O WASS2S_
↪notebooks.zip
```

1.2.1 Download module

Three types of data can be downloaded with wass2s:

- GCM data on seasonal time scales
- Reanalysis data
- Observational data (satellite data, products combining satellite and observational data)

For some data, for instance C3S, it requires creating an account, accepting the terms of use, and configuring an API key (**`CDS API key<https://hmandela.github.io/WAS_S2S_Training/s2s_data.html>`_**). Please refer also to the CDS documentation for more instructions on how to set up the API key. For more information on C3S seasonal data, browse the MetaData.

Download GCM data

The WAS_Download_Models method allows downloading seasonal forecast model data from various centers for specified variables, initialization months, lead times, and years.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- center_variable (list): List of center-variable identifiers, e.g., ["ECMWF_51.PRCP", "UKMO_604.TEMP"].
- month_of_initialization (int): Initialization month as an integer (1-12).
- lead_time (list): List of lead times in months.
- year_start_hindcast (int): Start year for hindcast data.
- year_end_hindcast (int): End year for hindcast data.
- area (list): Bounding box as [North, West, South, East] for clipping.
- year_forecast (int, optional): Forecast year if downloading forecast data. Defaults to None.
- ensemble_mean (str, optional): Can be "median", "mean", or None. Defaults to None.
- force_download (bool): If True, forces download even if file exists.

Available centers and variables:

- **Centers:** BOM_2, ECMWF_51, UKMO_604, UKMO_603, METEOFRANCE_8, METEOFRANCE_9, DWD_21, DWD_22, CMCC_35, NCEP_2, JMA_3, ECCC_4, ECCC_5, CFSV2_1, CMC1_1, CMC2_1, GFDL_1, NASA_1, NCAR_CCSM4_1, NMME_1
- **Variables:** PRCP, TEMP, TMAX, TMIN, UGRD10, VGRD10, SST, SLP, DSWR, DLWR, HUSS_1000, HUSS_925, HUSS_850, UGRD_1000, UGRD_925, UGRD_850, VGRD_1000, VGRD_925, VGRD_850

Note: Some models are part of the NMME (North American Multi-Model Ensemble) project. For more information, see the NMME documentation. If year_forecast is not specified, hindcast data is downloaded; otherwise, forecast data for the specified year is retrieved.

Example:

```
from wass2s import *

downloader = WAS_Download()

downloader.WAS_Download_Models(
    dir_to_save="/path/to/save",
    center_variable=["ECMWF_51.PRCP"],
    month_of_initialization="03",
    lead_time=["01", "02", "03"],
    year_start_hindcast=1993,
    year_end_hindcast=2016,
    area=[60, -180, -60, 180],
    force_download=False
)
```

Download daily GCM data

The WAS_Download_Models_Daily method allows downloading daily or sub-daily seasonal forecast model data from various centers for specified variables, initialization dates, lead times, and years.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- center_variable (list): List of center-variable identifiers, e.g., ["ECMWF_51.PRCP", "UKMO_604.TEMP"].
- month_of_initialization (int): Initialization month as an integer (1-12).
- day_of_initialization (int): Initialization day as an integer (1-31).
- leadtime_hour (list): List of lead times in hours, e.g., ["24", "48", ..., "5160"].
- year_start_hindcast (int): Start year for hindcast data.
- year_end_hindcast (int): End year for hindcast data.
- area (list): Bounding box as [North, West, South, East] for clipping.
- year_forecast (int, optional): Forecast year if downloading forecast data. Defaults to None.
- ensemble_mean (str, optional): Can be "mean", "median", or None. Defaults to None.
- force_download (bool): If True, forces download even if file exists.

Available centers and variables:

- **Centers:** ECMWF_51, UKMO_604, UKMO_603, METEOFRANCE_8, DWD_21, DWD_22, CMCC_35, NCEP_2, JMA_3, ECCC_4, ECCC_5
- **Variables:** PRCP, TEMP, TMAX, TMIN, UGRD10, VGRD10, SST, SLP, DSWR, DLWR, HUSS_1000, HUSS_925, HUSS_850, UGRD_1000, UGRD_925, UGRD_850, VGRD_1000, VGRD_925, VGRD_850

Example:

```
from wass2s import *

downloader = WAS_Download()
downloader.WAS_Download_Models_Daily(
    dir_to_save="/path/to/save",
    center_variable=["ECMWF_51.PRCP"],
    month_of_initialization="01",
```

(continues on next page)

(continued from previous page)

```
    day_of_initialization="01",
    leadtime_hour=["24", "48", "72"],
    year_start_hindcast=1993,
    year_end_hindcast=2016,
    area=[60, -180, -60, 180],
    force_download=False
)
```

Download reanalysis data

The WAS_Download_Reanalysis method downloads reanalysis data for specified center-variable combinations, years, and months, handling cross-year seasons.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- center_variable (list): List of center-variable identifiers, e.g., ["ERA5.PRCP", "MERRA2.TEMP"].
- year_start (int): Start year for the data to download.
- year_end (int): End year for the data to download.
- area (list): Bounding box as [North, West, South, East] for clipping.
- seas (list): List of month strings representing the season, e.g., ["11", "12", "01"] for NDJ.
- force_download (bool): If True, forces download even if file exists.
- run_avg (int): Number of months for running average (default=3).

Available centers and variables:

- **Centers:** ERA5, MERRA2, NOAA (for SST)
- **Variables:** PRCP, TEMP, TMAX, TMIN, UGRD10, VGRD10, SST, SLP, DSWR, DLWR, HUSS_1000, HUSS_925, HUSS_850, UGRD_1000, UGRD_925, UGRD_850, VGRD_1000, VGRD_925, VGRD_850

Example:

```
from wass2s import *

downloader = WAS_Download()
downloader.WAS_Download_Reanalysis(
    dir_to_save="/path/to/save",
    center_variable=["ERA5.PRCP"],
    year_start=1993,
    year_end=2016,
    area=[60, -180, -60, 180],
    seas=["11", "12", "01"],
    force_download=False
)
```

Download observational data

Observational data includes agro-meteorological indicators and satellite-based precipitation data like CHIRPS.

Agro-meteorological indicators

The WAS_Download_AgroIndicators method downloads agro-meteorological indicators for specified variables, years, and months, handling cross-year seasons.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- variables (list): List of shorthand variables, e.g., ["AGRO.PRCP", "AGRO.TMAX"].
- year_start (int): Start year for the data to download.
- year_end (int): End year for the data to download.
- area (list): Bounding box as [North, West, South, East] for clipping.
- seas (list): List of month strings representing the season, e.g., ["11", "12", "01"] for NDJ.
- force_download (bool): If True, forces download even if file exists.

Available variables:

- AGRO.PRCP: precipitation_flux
- AGRO.TMAX: 2m_temperature (24_hour_maximum)
- AGRO.TEMP: 2m_temperature (24_hour_mean)
- AGRO.TMIN: 2m_temperature (24_hour_minimum)

Example:

```
from wass2s import *

downloader = WAS_Download()
downloader.WAS_Download_AgroIndicators(
    dir_to_save="/path/to/save",
    variables=["AGRO.PRCP"],
    year_start=1993,
    year_end=2016,
    area=[60, -180, -60, 180],
    seas=["11", "12", "01"],
    force_download=False
)
```

Download daily agro-meteorological indicators

The WAS_Download_AgroIndicators_daily method downloads daily agro-meteorological indicators for specified variables and years.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- variables (list): List of shorthand variables, e.g., ["AGRO.PRCP", "AGRO.TMAX"].
- year_start (int): Start year for the data to download.
- year_end (int): End year for the data to download.
- area (list): Bounding box as [North, West, South, East] for clipping.
- force_download (bool): If True, forces download even if file exists.

Available variables:

- AGRO.PRCP: precipitation_flux
- AGRO.TMAX: 2m_temperature (24_hour_maximum)
- AGRO.TEMP: 2m_temperature (24_hour_mean)
- AGRO.TMIN: 2m_temperature (24_hour_minimum)

Example:

```
from wass2s import *

downloader = WAS_Download()
downloader.WAS_Download_AgroIndicators_daily(
    dir_to_save="/path/to/save",
    variables=["AGRO.PRCP"],
    year_start=1993,
    year_end=2016,
    area=[60, -180, -60, 180],
    force_download=False
)
```

CHIRPS precipitation data

The WAS_Download_CHIRPSv3 method downloads CHIRPS v3.0 monthly precipitation data for a specified cross-year season.

Parameters:

- dir_to_save (str): Directory to save the downloaded files.
- variables (list): List of variables, typically ["PRCP"].
- year_start (int): Start year for the data to download.
- year_end (int): End year for the data to download.
- area (list, optional): Bounding box as [North, West, South, East] for clipping.
- season_months (list): List of month strings representing the season, e.g., ["03", "04", "05"] for MAM.
- force_download (bool): If True, forces download even if file exists.

Note: CHIRPS data is available for land areas between 50°S and 50°N.

Example:

```
from wass2s import *

downloader = WAS_Download()
downloader.WAS_Download_CHIRPSv3(
    dir_to_save="/path/to/save",
    variables=["PRCP"],
    year_start=1993,
    year_end=2016,
    area=[15, -20, -5, 20],  # Example for Africa
    season_months=["03", "04", "05"],
    force_download=False
)
```

1.2.2 Processing Modules

The Processing modules provide tools for computing various climate indices or predictands from daily data, such as onset and cessation of the rainy season, dry and wet spells, number of rainy days, extreme precipitation indices, and heat wave indices. Additionally, it offers methods for merging or adjusting gridded data with station observations to correct biases.

These modules are divided into two main parts:

1. **Computing Predictands**: Classes for calculating different climate indices from daily data.
2. **Merging and Adjusting Data**: Classes for combining gridded data with station observations to improve accuracy.

Prerequisites

- **Dask**: Required for parallel processing in gridded data computations.
- **Data Formats**: Gridded data should be in xarray DataArray format with coordinates (T, Y, X). Station data should be in CDT format for daily data or CPT format for seasonal aggregation before merging.

Climate Data Tools (CDT): Format for daily data.

ID	ALLADA	APLAHOUE
LON	2.133333	1.666667
LAT	6.65	6.916667
DAILY/ELEV	92.0	153.0
19810101	0.0	0.0
19810102	0.0	0.0
19810103	0.0	0.0
19810104	0.0	0.0
19810105	0.0	0.0
19810106	0.0	0.0
19810107	0.0	0.0
19810108	0.0	0.0
19810109	0.0	0.0
19810110	0.0	0.0
...		

Climate Prediction Tools (CPT): Format for seasonal aggregation (used in climate prediction tools) before merging.

STATION	ABEO	ABUJ	ADEK
LAT	7.2	7.6	9.0
LON	3.3	5.2	7.2
1991	514.9	715.1	934.3
1992	503.6	736.4	714.6
1993	414.6	891.0	709.6
1994	345.6	1034.7	491.7
1995	492.2	837.6	938.8
...			

Computing Predictands

This section includes classes for computing various climate indices:

- WAS_compute_onset: Computes the onset of the rainy season.
- WAS_compute_cessation: Computes the cessation of the rainy season.
- WAS_compute_onset_dry_spell: Computes the longest dry spell after the onset.
- WAS_compute_cessation_dry_spell: Computes the longest dry spell in flourishing period.
- WAS_count_wet_spells: Computes the number of wet spells between onset and cessation.
- WAS_count_dry_spells: Computes the number of dry spells between onset and cessation.
- WAS_count_rainy_days: Computes the number of rainy days between onset and cessation.
- WAS_r95_99p: Computes extreme precipitation indices R95p and R99p.
- WAS_compute_HWSDI: Computes the Heat Wave Severity Duration Index.

Each class has methods for computing the index from gridded data (compute) and, where applicable, from station data in CDT format (compute_insitu).

Onset Computation

The WAS_compute_onset class computes the onset of the rainy season based on user-defined or default criteria for different zones.

Initialization

- __init__(self, user_criteria=None): Initializes the class with user-defined criteria. If not provided, default criteria are used.
- Dictionaries onset_criteria, cessation_criteria, onset_dryspell_criteria, cessation_dryspell_criteria show how to define the criteria for onset, cessation, onset dry spell and cessation dry spell computations.

Methods

- compute(self, daily_data, nb_cores): Computes onset dates for gridded daily rainfall data. * daily_data: xarray DataArray with daily rainfall data (coords: T, Y, X). * nb_cores: Number of CPU cores for parallel processing. * Returns: xarray DataArray with onset dates.
- compute_insitu(self, daily_df): Computes onset dates for station data in CDT format. * daily_df: pandas DataFrame in CDT format. * Returns: pandas DataFrame in CPT format with onset dates.

Criteria Dictionary

The criteria dictionary defines parameters for onset computation:

```
{
  0: {"zone_name": "Sahel100_0mm", "start_search": "06-01", "cumulative": 10, "number_dry_days
↪": 25, "thrd_rain_day": 0.85, "end_search": "08-30"},
  1: {"zone_name": "Sahel200_100mm", "start_search": "05-15", "cumulative": 15, "number_dry_days
↪": 25, "thrd_rain_day": 0.85, "end_search": "08-15"},
  ...
}
```

- zone_name: Name of the zone.
- start_search: Start date for searching the onset (e.g., “06-01”).
- cumulative: Cumulative rainfall threshold (mm).
- number_dry_days: Maximum number of dry days allowed after onset.

- thrd_rain_day: Rainfall threshold to consider a day as rainy (mm).
- end_search: End date for searching the onset.

Example

```
from wass2s import *
# Download daily rainfall data
downloader = WAS_Download()
downloader.WAS_Download_AgroIndicators_daily(
    dir_to_save="/path/to/save",
    variables=["AGRO.PRCP"],
    year_start=1993,
    year_end=2016,
    area=[60, -180, -60, 180],
    force_download=False
)

# Load daily rainfall data
rainfall = prepare_predictand(dir_to_save, variables, year_start, year_end, daily=True, ds=False)
## NB: prepare_predictand is a utility function that loads the data and prepares it for the computation␣
↪of the predictand.
## ds is set to False because the data will be loaded as dataarray.

# Print predefined  onset criteria
onset_criteria
# Define user criteria
user_criteria = onset_criteria
# adjust user criteria
user_criteria[0]["start_search"] = "06-15"
user_criteria[1]["end_search"] = "09-01"
# Compute onset
was_onset = WAS_compute_onset(user_criteria)
onset = was_onset.compute(daily_data=rainfall, nb_cores=4)
# Plot the mean onset date to check the results
plot_date(onset.mean(dim='T'))
```

Cessation Computation

The WAS_compute_cessation class computes the cessation of the rainy season based on soil moisture balance criteria.

- Similar initialization and methods as WAS_compute_onset with criteria including: * date_dry_soil: Date when soil is assumed dry (e.g., “01-01”). * ETP: Evapotranspiration rate (mm/day). * Cap_ret_maxi: Maximum soil water retention capacity (mm).

Dry Spell Computation

The WAS_compute_onset_dry_spell class computes the longest dry spell after the onset.

- Includes an additional nbjour parameter in the criteria for the number of days to check after onset.

The WAS_compute_cessation_dry_spell class computes the longest dry spell in flourishing period.

- Includes an additional nbjour parameter in the criteria for the number of days to check after cessation.

The WAS_count_dry_spells class computes the number of dry spells between onset and cessation. Requires onset and cessation dates as inputs.

Wet Spell Computation

The WAS_count_wet_spells class computes the number of wet spells between onset and cessation. Requires onset and cessation dates as inputs.

Rainy Days Computation

The WAS_count_rainy_days class computes the number of rainy days between onset and cessation. Requires onset and cessation dates as inputs.

Extreme Precipitation Indices

The WAS_r95_99p class computes R95p and R99p indices. Initialization with a base period (e.g., slice("1991-01-01", "2020-12-31")) and optional season (list of months).

- Methods: * compute_r95p and compute_r99p for gridded data. * compute_insitu_r95p and compute_insitu_r99p for station data.

Heat Wave Indices

The WAS_compute_HWSDI class computes the Heat Wave Severity Duration Index. Computes TXin90 (90th percentile of daily max temperature) and counts heatwave days with at least 6 consecutive hot days.

Merging and Adjusting Data

The WAS_Merging class provides methods for merging gridded data with station observations to adjust for biases.

Initialization

- __init__(self, df, da, date_month_day="08-01"): Initializes with station data DataFrame (CPT format), gridded data DataArray, and a date string.

Methods

- simple_bias_adjustment(self, missing_value=-999.0, do_cross_validation=False): Adjusts gridded data using kriging of residuals.
- regression_kriging(self, missing_value=-999.0, do_cross_validation=False): Uses linear regression followed by kriging of residuals.
- neural_network_kriging(self, missing_value=-999.0, do_cross_validation=False): Uses a neural network followed by kriging of residuals.
- multiplicative_bias(self, missing_value=-999.0, do_cross_validation=False): Applies a multiplicative bias correction.

Each method returns the adjusted gridded data as an xarray DataArray and optionally cross-validation results as a DataFrame.

- plot_merging_comparaison(self, df_Obs, da_estimated, da_corrected, missing_value=-999.0): Visualizes the comparison between observations, original estimates, and corrected data.

Example: Merging Onset with Station Observations

```
# Load station onset data in CPT format
cpt_input_file_path = "./path/to/cpt_file.csv"
df = pd.read_csv(cpt_input_file_path, na_values=-999.0, encoding="latin1")

# Filter for relevant years and stations
year_start, year_end = 1981, 2020  # Example years
onset_df = df[(df['STATION'] == 'LAT') | (df['STATION'] == 'LON') |
              (pd.to_numeric(df['STATION'], errors='coerce').between(year_start, year_end))]

# Verify station network
```

(continues on next page)

(continued from previous page)

```
verify_station_network(onset_df, area)
## NB: verify_station_network is a utility function that verifies the station network. area is the extent␣
↪of the gridded onset domain.

# Instantiate WAS_Merging
data_merger = WAS_Merging(onset_df, onset, date_month_day='02-01')
## NB: date_month_day is set to '02-01' because the onset start_search criteria is set to the month of␣
↪February.
## Important to verify the T dimension in the gridded onset computed. the month and day must match␣
↪the date_month_day.

# Perform simple bias adjustment
onset_adjusted, _ = data_merger.simple_bias_adjustment(do_cross_validation=False)

# Plot comparison
data_merger.plot_merging_comparaison(onset_df, onset, onset_adjusted)
## NB: plot_merging_comparaison is a utility function that plots the comparison between the station␣
↪onset, the gridded onset and the adjusted onset.
```

1.2.3 Quantifying uncertainty via cross-validation

Cross-validation schemes are used to assess model performance and to quantify uncertainty. *wass2s* uses a cross-validation scheme that splits the data into training, omit, and test periods. The scheme is a variation of the *K-Fold* cross-validation scheme, but it is tailored for time series data throughout *CustomTimeSeriesSplit* and *WAS_Cross_Validator* class. The scheme is illustrated in the figure below (Figure 1).

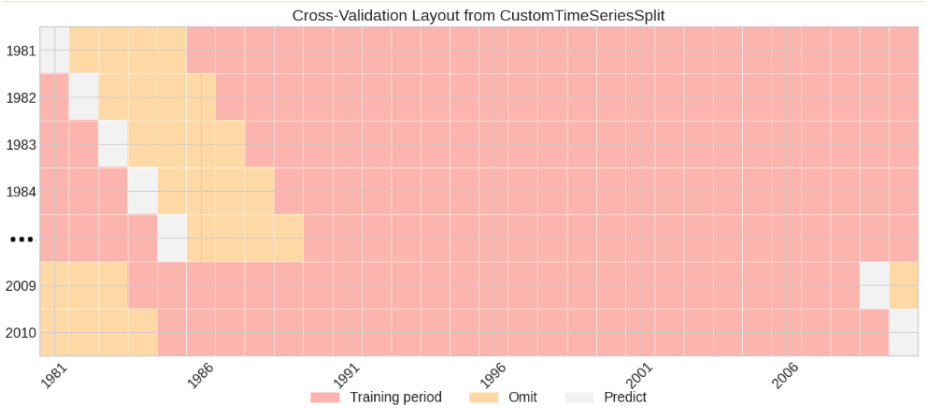

Fig. 1: Figure 1: Cross-validation scheme used in wass2s

The figure shows how we split our data (1981–2010) to validate the model. Each row is a "fold" or a test run.

- **Pink (Training)**: Years we use to train the model. For example, in the first row, we train on 1986–2010.
- **Yellow (Omit)**: A buffer years we skip to avoid cheating. Climate data has patterns over time, so we don't want to train on a years right after/before the one we're predicting, which would make the model look better than it really is. In this case we've omitted four years (in the first row, we skip 1982-1985).
- **White (Predict)**: The year we predict. In the first row, we predict 1981.

CustomTimeSeriesSplit

A custom splitter for time series data that accounts for temporal dependencies.

Initialization

- *n_splits*: Number of splits for cross-validation.

Methods

- *split*: Generates indices for training and test sets, omitting a specified number of samples after the test index.
- *get_n_splits*: Returns the number of splits.

WAS_Cross_Validator

A wrapper class that uses the custom splitter to perform cross-validation with various models.

Initialization

- *n_splits*: Number of splits for cross-validation.
- *nb_omit*: Number of samples to omit from training after the test index.

Methods

- *get_model_params*: Retrieves parameters for the model's *compute_model* method.
- *cross_validate*: Performs cross-validation and computes deterministic hindcast and tercile probabilities.

Example Usage

```
from wass2s.was_cross_validate import WAS_Cross_Validator

# Initialize the cross-validator
cv = WAS_Cross_Validator(n_splits=30, nb_omit=4)
```

A better example will be provided in the next sections.

Estimating Prediction Uncertainty

The cross-validation makes out-of-sample predictions for each fold's prediction period, and errors are calculated by comparing predictions to actual values. These errors are collected across all folds. Running the statistical models—e.g. multiple linear regression—yields the most likely value of the predictand (best-guess) for the coming season. Because seasonal outlooks are inherently probabilistic, we must go beyond this single best-guess and quantify the likelihood of other possible outcomes. wass2s does so by analysing the cross-validation errors described earlier. The method explicitly takes the statistical distribution of the predictand into account. If, for instance, the predictand is approximately Gaussian, we assume the predicted values follow a normal distribution whose mean is the single best-guess and whose variance equals the cross-validated error variance. Comparing that forecast probability-density function with the climatological density (see the example in Figure 2) lets us integrate the areas that fall below-normal (values below the 1st tercile), near-normal (values between the 1st and 3rd terciles), and above-normal (values above the 3rd tercile). These integrals are the tercile probabilities ultimately delivered to users.

1.2.4 Models Modules

The Models modules provide a comprehensive suite of statistical and machine learning models for climate prediction, including linear models, EOF-based models, canonical correlation analysis (CCA), analog methods, and multi-model ensemble (MME) techniques. These models are designed to handle both deterministic and probabilistic forecasts, with support for hyperparameter tuning. Models are evaluated using cross-validation schemes.

The models modules are organized into several classes, each implementing a specific type of model:

1. **Machine Learning Models**: This includes linear models such as multiple linear regression, logistic regression and regularized models like ridge, lasso, elastic-net. Additionally, more advanced models are available, including support vector regression, random forests, XGBoost, and neural networks.
2. **EOF and PCR Models**: For dimensionality reduction and regression using principal components.

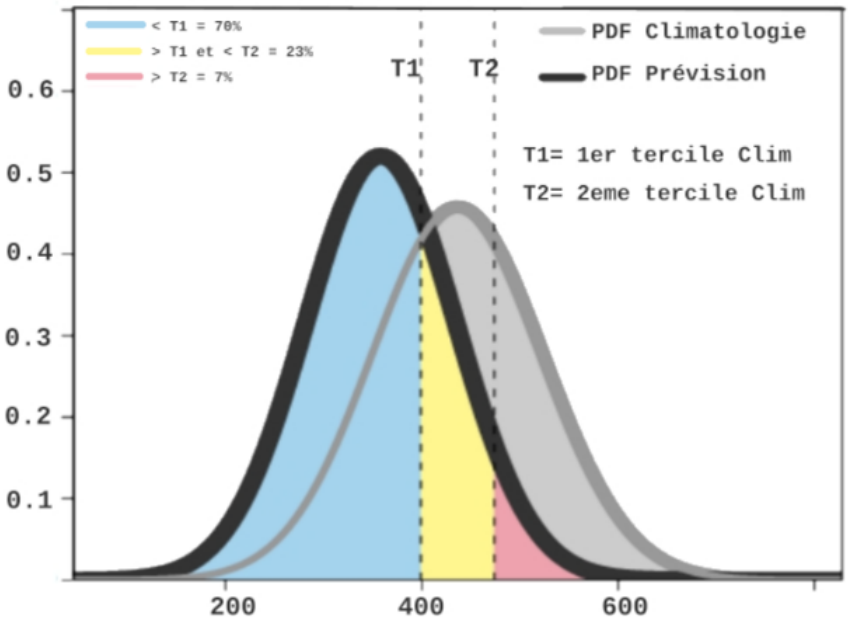

Fig. 2: Figure 2: Generation of probabilistic forecasts

3. **CCA Models**: For identifying relationships between two multivariate datasets.
4. **Analog Methods**: For finding historical analogs to current conditions.
5. **Multi-Model Ensemble (MME) Techniques**: For combining predictions from multiple models.

Machine Learning Models

The available models are:

- *WAS_LinearRegression_Model*: Standard Multiple Linear Regression.
- *WAS_Ridge_Model*: Ridge regression with L2 regularization.
- *WAS_Lasso_Model*: Lasso regression with L1 regularization.
- *WAS_LassoLars_Model*: Lasso regression using the LARS algorithm.
- *WAS_ElasticNet_Model*: Elastic net regression combining L1 and L2 regularization.
- *WAS_LogisticRegression_Model*: Logistic regression for classification.
- *WAS_SVR*: Support vector regression.
- *WAS_PolynomialRegression*: Polynomial regression.
- *WAS_PoissonRegression*: Poisson regression.
- *WAS_RandomForest_XGBoost_ML_Stacking*: Random forest and XGBoost regression with stacking.
- *WAS_MLP*: Multi-Layer Perceptron regression.
- *WAS_RandomForest_XGBoost_Stacking_MLP*: Random forest, XGBoost, and MLP regression with stacking.
- *WAS_Stacking_Ridge*: Random forest, XGBoost, MLP, and Ridge regression with stacking.

Except for *WAS_LogisticRegression_Model*, each model class includes methods for:

- *compute_model*: Training the model and making predictions.

- *compute_prob*: Computing tercile probabilities for the predictions.
- *forecast*: Making forecasts for new data.

EOF and PCR Models

The *was_eof.py* and *was_pcr.py* modules provide classes for EOF analysis and Principal Component Regression (PCR), with support for multiple EOF zones:

- *WAS_EOF*: Performs EOF analysis with options for cosine latitude weighting, standardization, and L2 normalization.
- *WAS_PCR*: Combines EOF analysis with a regression model for prediction, supporting multiple EOF zones.

WAS_EOF

Initialization

- *n_modes*: Number of EOF modes to retain.
- *use_coslat*: Apply cosine latitude weighting (default: True).
- *standardize*: Standardize the input data (default: False).
- *opti_explained_variance*: Target cumulative explained variance to determine modes.
- *L2norm*: Normalize components and scores to have L2 norm (default: True).

Methods

- *fit*: Fits the EOF model to the data, supporting multiple zones by applying EOF analysis to the entire dataset.
- *transform*: Projects new data onto the EOF modes.
- *inverse_transform*: Reconstructs data from principal components (PCs).
- *plot_EOF*: Plots the EOF spatial patterns with explained variance.

WAS_PCR

Initialization

- *regression_model*: The regression model (e.g., *WAS_Ridge_Model*) to use with PCs.
- *n_modes*: Number of EOF modes to retain.
- *use_coslat*: Apply cosine latitude weighting (default: True).
- *standardize*: Standardize the input data (default: False).
- *opti_explained_variance*: Target cumulative explained variance.
- *L2norm*: Normalize EOF components and scores (default: True).

Methods

- *compute_model*: Fits the EOF model, transforms data to PCs, and applies the regression model.
- *compute_prob*: Computes tercile probabilities using the regression model.
- *forecast*: Makes forecasts using EOF-transformed data.

Example Usage: Seasonal Forecasting Based on Observational Data

```
from wass2s import *
## Define the directory to save the data
dir_to_save_reanalysis = "/path/to/save_reanalysis"
dir_to_save_agroindicators = "/path/to/save_agroindicators"
```

(continues on next page)

(continued from previous page)

```
## Define the climatology  year range and the season
clim_year_start = 1991
clim_year_end = 2020
seas_reanalysis = ["01", "02", "03"]
seas_agroindicators = ["05", "06", "07"]

## Define the variables to download
variables = ["AGRO.PRCP"]

## Define the center and the predictor variables
center_variable = ["ERA5.SST"]:

## Define the extent for reanalysis
extent = [45, -180, -45, 180] # [North, West, South, East]

## Define the extent for Observation
extent_obs = [30, -25, 0, 30] # [North, West, South, East]

## Download the predictors and the predictand
downloader = WAS_Download()

## Download the predictors
downloader.WAS_Download_Reanalysis(
    dir_to_save=dir_to_save_reanalysis,
    center_variable=center_variable,
    year_start=1991,
    year_end=2025,
    area=extent,
    seas=seas_reanalysis,
    force_download=False
)

## Download the predictand
downloader.WAS_Download_AgroIndicators(
    dir_to_save=dir_to_save_agroindicators,
    variables=["AGRO.PRCP"],
    year_start=1991,
    year_end=2024,
    area=extent_obs,
    seas=seas_agroindicators,
    force_download=False
)
```

Case 1: Used SST index as a predictor

```
# Prepare predictand and predictors
predictand = prepare_predictand(dir_to_save_agroindicators, variables, year_start, year_end, seas_
↪agroindicators, ds=False, daily=False)

# Prepare predictors
## Print available SST indices
```

(continues on next page)

(continued from previous page)

```
print(list(sst_indices_name.keys()))

## Choose yours
sst_index_name = ['NINO34','TNA', 'TSA', 'DMI']

## Plot the SST index zone
plot_map([extent[1],extent[3],extent[2],extent[0]], sst_indices = sst_index_name, title="Index Zone",fig_
↪size=(7,4))

## Compute the SST indices
predictors = compute_sst_indices(dir_to_save_reanalysis, sst_index_name, center_variable[0], year_
↪start, year_end, seas_reanalysis)

## Compute variance inflation factor to see multicolinearity between predictors

vif_data = pd.DataFrame()
vif_data["feature"] = predictors.to_dataframe().columns
vif_data["VIF"] = [VIF(predictors.to_dataframe(), i) for i in range(predictors.to_dataframe().shape[1])]
## Print VIF values
print(vif_data)

## Set a threshold for VIF
vif_threshold = 5
# Remove features with VIF greater than the threshold
low_vif_predictors = vif_data[vif_data["VIF"] < vif_threshold]["feature"].tolist()
filtered_predictors = predictors[low_vif_predictors].to_array()
filtered_predictors = filtered_predictors.rename({"variable": "features"}).transpose('T', 'features')

# Initialize the model class
model = WAS_LinearRegression_Model(nb_cores=2, dist_method="lognormal")
# Assuming predictand follows a lognormal distribution. otherwise, normal, student-t or gamma are␣
↪available. used dist_method="normal" or dist_method="t" or dist_method="gamma".

# Perform cross-validation
was_cv = WAS_Cross_Validator(n_splits=len(predictand.get_index("T")), nb_omit=2)
hindcast_det, hindcast_prob = was_cv.cross_validate(model, predictand, filtered_
↪predictorsisel(T=slice(None,-1)), clim_year_start, clim_year_end)
# clim_year_start and clim_year_end are the years used to compute the climatology.

# Initialize the model class
model = WAS_Ridge_Model(n_clusters=6, alpha_range=np.logspace(-4, 0.1, 20), nb_cores = 2)

# Compute alpha parameters
alpha, clusters = model.compute_hyperparameters(predictand, filtered_predictors)

# Perform cross-validation
was_cv = WAS_Cross_Validator(n_splits=len(predictand.get_index("T")), nb_omit=2)
hindcast_det_Ridge, hindcast_prob_Ridge = was_cv.cross_validate(model, predictand, filtered_
↪predictors.isel(T=slice(None,-1)), clim_year_start, clim_year_end, alpha=alpha)

# Make a forecast
forecast_det_Ridge, forecast_prob_Ridge = model.forecast(predictand, clim_year_start, clim_year_
```

(continues on next page)

(continued from previous page)

```
↪end, filtered_predictors.isel(T=slice(None,-1)), hindcast_det_Ridge, filtered_predictors.isel(T=[-1]),␣
↪alpha=alpha, l1_ratio=l1_ratio)
```

Case 2: Used PCRs as a predictor

```
# Set your own zones ( zones not available in built-in)
# define zone as dict : {'zone_name_key': ('Explicit_Zone_name', lon_min, lon_max, lat_min, lat_
↪max)}
zones_for_PCR = {'A': ('A', -150, 150, -45, 45)}

# Set number of modes
n_modes = 6

# ElasticNet hyperparameters range
alpha_range = np.logspace(-4, 0.1, 20)
l1_ratio_range = [0.5, 0.9999]

# Initialize the model class
model = WAS_PCR_Model(n_clusters=6, alpha_range=np.logspace(-4, 0.1, 20), nb_cores = 2)
plot_map([extent[1],extent[3],extent[2],extent[0]], sst_indices = zones_for_PCR, title="Predictors Area",
↪fig_size=(8,6))

# Retrieve predictor data for the defined zone
predictor = retrieve_single_zone_for_PCR(dir_to_save_Reanalysis, zones_for_PCR, variables_
↪reanalysis[0], year_start, year_end, season, clim_year_start, clim_year_end)

# Load WAS_EOF Class
eof_model = WAS_EOF(n_modes=n_modes, use_coslat=True, standardize=True)

# Load predictor, compute EOFs and retrieve component, scores and explained variances
s_eofs, s_pcs, s_expvar, _ = eof_model.fit(predictor, dim="T",  clim_year_start=clim_year_start,␣
↪clim_year_end=clim_year_end)

# Plot EOFs and explained variances
eof_model.plot_EOF(s_eofs, s_expvar)

# Perform Cross-validation with elastic-net

## Load class for model
regression_model = WAS_ElasticNet_Model(alpha_range = alpha_range, l1_ratio_range = l1_ratio_
↪range, nb_cores = 2, dist_method="lognormal")
pcr_model = WAS_PCR(regression_model=regression_model, n_modes=n_modes, standardize=False)

## Compute alpha parameters
alpha, l1_ratio, clusters = regression_model.compute_hyperparameters(predictand, s_pcs.
↪isel(T=slice(None,-1)).rename({"mode": "features"}).transpose('T', 'features'))
## Perform cross-validation
was_cv = WAS_Cross_Validator(n_splits=len(predictand.get_index("T")), nb_omit=2)
hindcast_det, hindcast_prob = was_cv.cross_validate(pcr_model, predictand, s_pcs.isel(T=slice(None,-
↪1)).rename({"mode": "features"}).transpose('T', 'features'), clim_year_start, clim_year_end,␣
↪alpha=alpha, l1_ratio=l1_ratio)
```

CCA Models

The *was_cca.py* module provides classes for Canonical Correlation Analysis (CCA):

- *WAS_CCA*: Performs CCA to identify relationships between two multivariate datasets.

Initialization

- *n_modes*: Number of CCA modes to retain.
- *n_pca_modes*: Number of PCA modes to use for dimensionality reduction.
- *dist_method*: distribution method for probability computations.

Methods

- *compute_model*: Fits the CCA model and makes predictions.
- *compute_prob*: Computes tercile probabilities for the predictions.

Example Usage: Recalibrating Seasonal Forecast Outputs from Global Climate Models (GCMs)

Analog Forecasting Methods

The *was_analog.py* module provides the *WAS_Analog* class for analog-based forecasting using various techniques to identify historical analogs to current conditions for prediction, particularly for seasonal rainfall forecasts using sea surface temperature (SST) data.

Initialization Parameters

- dir_to_save (str): Directory path to save downloaded and processed data files.
- year_start (int): Starting year for historical data.
- year_forecast (int): Target forecast year.
- reanalysis_name (str): Reanalysis dataset name (e.g., "ERA5.SST" or "NOAA.SST").
- model_name (str): Forecast model name (e.g., "ECMWF_51.SST").
- method_analog (str, default="som"): Analog method to use ("som", "cor_based", "pca_based").
- best_prcp_models (list, optional): List of best precipitation models. Default is None.
- month_of_initialization (int, optional): Forecast initialization month. Default is None (uses current month).
- lead_time (list, optional): Lead times in months. Default is None (uses [1, 2, 3, 4, 5]).
- ensemble_mean (str, default="mean"): Ensemble mean method ("mean" or "median").
- clim_year_start (int, optional): Start year for climatology period.
- clim_year_end (int, optional): End year for climatology period.
- define_extent (tuple, optional): Bounding box as (lon_min, lon_max, lat_min, lat_max) for regional analysis.
- index_compute (list, optional): Climate indices to compute (e.g., ["NINO34", "DMI"]).
- some_grid_size (tuple, default=(None, None)): SOM grid dimensions (rows, cols); None uses automatic sizing.
- some_learning_rate (float, default=0.5): Learning rate for SOM training.
- some_neighborhood_function (str, default="gaussian"): Neighborhood function for SOM ("gaussian", etc.).
- some_sigma (float, default=1.0): Initial neighborhood radius for SOM.
- dist_method (str, default="gamma"): Probability method ("gamma", "t", "normal", "lognormal", "nonparam").

Key Methods

- download_sst_reanalysis(): Downloads and processes SST reanalysis data from the specified center for the given years and area.
- download_models(): Downloads seasonal forecast model data for the specified model, initialization month, and lead times.
- standardize_timeseries(): Standardizes time series data over a specified climatology period.
- calc_index(): Computes specified climate indices (e.g., NINO34, DMI) from SST data.
- compute_model(): Identifies historical analogs using the specified method and computes deterministic forecasts.
- compute_prob(): Calculates tercile probabilities (Below Normal, Near Normal, Above Normal) using the specified distribution method.
- forecast(): Generates deterministic and probabilistic forecasts for the target year, returning processed SST data, similar years, deterministic forecast, and probabilistic forecast.
- composite_plot(): Creates composite plots of forecast results, optionally including the predictor (SST) visualization.

Example Usage

Basic analog forecast setup:

```
from wass2s.was_analog import WAS_Analog

# Initialize analog model
analog_model = WAS_Analog(
    dir_to_save="./s2s_data/analog",
    year_start=1990,
    year_forecast=2025,
    reanalysis_name="NOAA.SST",
    model_name="ECMWF_51.SST",
    method_analog="som",
    month_of_initialization=3,
    clim_year_start=1991,
    clim_year_end=2020,
    define_extent=(-180, 180, -45, 45),
    index_compute=["NINO34", "DMI"],
    dist_method="gamma"
)

# Download and process data
sst_hist, sst_for = analog_model.download_and_process()

# Generate forecast
ddd, similar_years, forecast_det, forecast_prob = analog_model.forecast(
    predictant=rainfall_data,
    clim_year_start=1991,
    clim_year_end=2020,
    hindcast_det=hindcast_data,
    forecast_year=2025
)

# Create composite plot
similar_years = analog_model.composite_plot(
```

(continues on next page)

(continued from previous page)

```
    predictant=rainfall_data,
    clim_year_start=1991,
    clim_year_end=2020,
    hindcast_det=hindcast_data,
    plot_predictor=True
)
```

Cross-Validation Example

```
from wass2s.was_analog import WAS_Cross_Validator

# Perform cross-validation
was_analog_cv = WAS_Cross_Validator(n_splits=len(rainfall.get_index("T")), nb_omit=2)
hindcast_analog_det, hindcast_analog_prob = was_analog_cv.cross_validate(
    analog_model,
    rainfall,
    clim_year_start=1991,
    clim_year_end=2020
)

# Generate forecast using cross-validated hindcast
ddd, similar_years, forecast_det, forecast_prob = analog_model.forecast(
    predictant=rainfall,
    clim_year_start=1991,
    clim_year_end=2020,
    hindcast_det=hindcast_analog_det,
    forecast_year=2025
)
```

Note

Ensure *WAS_Cross_Validator* is correctly imported from the *wass2s.was_analog* module and that the *rainfall* variable is an xarray DataArray with appropriate dimensions (T, Y, X).

1.2.5 Verification Module

The Verification module provides tools for evaluating the performance of climate forecasts using a variety of deterministic, probabilistic, and ensemble-based metrics. It is implemented in the *was_verification.py* module and leverages the *WAS_Verification* class to compute metrics such as Kling-Gupta Efficiency (KGE), Pearson Correlation, Ranked Probability Skill Score (RPSS), and Continuous Ranked Probability Score (CRPS). The module also includes visualization utilities for plotting scores, reliability diagrams, and ROC curves.

This module is designed to work with gridded climate data, typically stored in *xarray* DataArrays, and supports parallel computation using *dask* for efficiency with large datasets.

The *WAS_Verification* class is the core of the Verification module, providing methods to compute and visualize various performance metrics for climate forecasts.

Initialization

```
from wass2s.was_verification import WAS_Verification
```

(continues on next page)

(continued from previous page)

```
# Initialize with a distribution method for probabilistic forecasts
verifier = WAS_Verification(dist_method="gamma")
```

Parameters

- *dist_method*: Specifies the distribution method for computing tercile probabilities. Options include: - *"t"*: Student's t-based method. - *"gamma"*: Gamma distribution-based method (default). - *"normal"*: Normal distribution-based method. - *"lognormal"*: Lognormal distribution-based method. - *"weibull_min"*: Weibull minimum distribution-based method. - *"nonparam"*: Non-parametric method using historical errors.

Available Metrics

The class defines a dictionary of scoring metrics with metadata, including:

- **Deterministic Metrics**: - *KGE*: Kling-Gupta Efficiency (-1 to 1). - *Pearson*: Pearson Correlation Coefficient (-1 to 1). - *IOA*: Index of Agreement (0 to 1). - *MAE*: Mean Absolute Error (0 to 100). - *RMSE*: Root Mean Square Error (0 to 100). - *NSE*: Nash-Sutcliffe Efficiency (None to 1). - *TAYLOR_DIAGRAM*: Taylor Diagram (visualization).
- **Probabilistic Metrics**: - *GROC*: Generalized Receiver Operating Characteristic (0 to 1). - *RPSS*: Ranked Probability Skill Score (-1 to 1). - *IGS*: Ignorance Score (0 to None). - *RES*: Resolution Score (0 to None). - *REL*: Reliability Score (None to None). - *RELIABILITY_DIAGRAM*: Reliability Diagram (visualization). - *ROC_CURVE*: Receiver Operating Characteristic Curve (visualization).
- **Ensemble Metrics**: - *CRPS*: Continuous Ranked Probability Score (0 to 100).

Metadata Access

```
metadata = verifier.get_scores_metadata()
```

This returns a dictionary containing the name, range, type, colormap, and computation function for each metric.

Deterministic Metrics

Deterministic metrics evaluate the performance of point forecasts against observations. They are computed using the *compute_deterministic_score* method, which applies a scoring function over *xarray* DataArrays.

Example Usage

```
# Compute Pearson Correlation
pearson_score = verifier.compute_deterministic_score(
    verifier.pearson_corr, obs_data, model_data
)

# Plot the score
verifier.plot_model_score(pearson_score, "Pearson", dir_save_score="./scores", figure_name=
↪"Pearson_Score")
```

Key Methods

- *kling_gupta_efficiency*: Computes KGE, balancing correlation, bias, and variability.
- *pearson_corr*: Computes Pearson Correlation Coefficient.
- *index_of_agreement*: Computes IOA, measuring agreement between predictions and observations.
- *mean_absolute_error*: Computes MAE, the average absolute difference.
- *root_mean_square_error*: Computes RMSE, the square root of mean squared differences.

- *nash_sutcliffe_efficiency*: Computes NSE, comparing prediction errors to the mean of observations.
- *taylor_diagram*: Placeholder for Taylor Diagram visualization (to be implemented).

Plotting

The *plot_model_score* method visualizes deterministic scores on a map using *cartopy*.

```
verifier.plot_model_score(score_result, "KGE", dir_save_score="./scores", figure_name="KGE_Model
↪")
```

The *plot_models_score* method plots multiple model scores in a grid.

```
model_metrics = {
    "model1": score_result1,
    "model2": score_result2
}
verifier.plot_models_score(model_metrics, "Pearson", dir_save_score="./scores")
```

Probabilistic Metrics

Probabilistic metrics evaluate the performance of forecasts that provide probabilities for tercile categories (below-normal, near-normal, above-normal). These are computed using the *compute_probabilistic_score* method.

Example Usage

```
# Compute tercile probabilities
proba_forecast = verifier.gcm_compute_prob(obs_data, clim_year_start=1981, clim_year_end=2010,␣
↪hindcast_det=model_data)

# Compute RPSS
rpss_score = verifier.compute_probabilistic_score(
    verifier.calculate_rpss, obs_data, proba_forecast, clim_year_start=1981, clim_year_end=2010
)
```

Key Methods

- *classify*: Classifies data into terciles based on climatology.
- *compute_class*: Computes tercile class labels for observations.
- *calculate_groc*: Computes GROC, averaging AUC across tercile categories.
- *calculate_rpss*: Computes RPSS, comparing forecast probabilities to climatology.
- *ignorance_score*: Computes Ignorance Score per Weijs (2010).
- *resolution_score_grid*: Computes Resolution Score, measuring how forecasts differ from climatology.
- *reliability_score_grid*: Computes Reliability Score, assessing forecast calibration.
- *reliability_diagram*: Plots Reliability Diagrams for each tercile category.
- *plot_roc_curves*: Plots ROC Curves with confidence intervals for each tercile.

Visualization

Reliability Diagrams and ROC Curves are generated for probabilistic forecasts.

```
# Plot Reliability Diagram
verifier.reliability_diagram(
    modelname="Model1", dir_to_save_score="./scores", y_true=obs_data, y_probs=proba_forecast,
    clim_year_start=1981, clim_year_end=2010
)

# Plot ROC Curves with 95% confidence intervals
verifier.plot_roc_curves(
    modelname="Model1", dir_to_save_score="./scores", y_true=obs_data, y_probs=proba_forecast,
    clim_year_start=1981, clim_year_end=2010, n_bootstraps=1000, ci=0.95
)
```

Ensemble Metrics

Ensemble metrics evaluate forecasts with multiple members, such as those from GCMs. The primary metric is CRPS, computed using *xskillscore*.

Example Usage

```
# Compute CRPS for ensemble forecast
crps_score = verifier.compute_crps(obs_data, model_data, member_dim='number', dim="T")
```

Key Methods

- *compute_crps*: Computes CRPS for ensemble forecasts, measuring the difference between predicted and observed distributions.

Tercile Probability Computation

The module provides multiple methods to compute tercile probabilities for probabilistic forecasts, based on different distributional assumptions.

Key Methods

- *calculate_tercile_probabilities*: Uses Student's t-distribution.
- *calculate_tercile_probabilities_gamma*: Uses Gamma distribution.
- *calculate_tercile_probabilities_normal*: Uses Normal distribution.
- *calculate_tercile_probabilities_lognormal*: Uses Lognormal distribution.
- *calculate_tercile_probabilities_weibull_min*: Uses Weibull minimum distribution.
- *calculate_tercile_probabilities_nonparametric*: Uses historical errors for a non-parametric approach.

Example Usage

```
# Compute probabilities using Gamma distribution
hindcast_prob = verifier.gcm_compute_prob(
    Predictant=obs_data, clim_year_start=1981, clim_year_end=2010, hindcast_det=model_data
)
```

The *gcm_compute_prob* method selects the appropriate distribution based on the *dist_method* parameter.

GCM Validation

The module includes methods to validate General Circulation Model (GCM) forecasts against observations, supporting both deterministic and probabilistic metrics.

Key Methods

- *gcm_validation_compute*: Validates GCM forecasts for multiple models, computing specified metrics.
- *weighted_gcm_forecasts*: Combines forecasts from multiple models using weights based on a performance metric (e.g., GROC).

Example Usage

```
# Validate GCM forecasts
models_files_path = {
    "model1": "path/to/model1.nc",
    "model2": "path/to/model2.nc"
}
x_metric = verifier.gcm_validation_compute(
    models_files_path=models_files_path, Obs=obs_data, score="Pearson",
    month_of_initialization=3, clim_year_start=1981, clim_year_end=2010,
    dir_to_save_roc_reliability="./scores", lead_time=[1]
)

# Compute weighted GCM forecasts
hindcast_det, hindcast_prob, forecast_prob = verifier.weighted_gcm_forecasts(
    Obs=obs_data, best_models={"model1_MarIc_JFM_1": score1}, scores={"GROC": x_metric},
    lead_time=[1], model_dir="./models", clim_year_start=1981, clim_year_end=2010, variable=
↪"PRCP"
)
```

Annual Year Validation

The module provides utilities to validate forecasts for a specific year, including ratio-to-average classification and RPSS computation.

Key Methods

- *ratio_to_average*: Classifies forecast data relative to the climatological mean into categories (e.g., Well Above Average, Near Average).
- *compute_one_year_rpss*: Computes RPSS for a specific year and visualizes it on a map.

Example Usage

```
# Classify ratio to average for a specific year
verifier.ratio_to_average(predictant=obs_data, clim_year_start=1981, clim_year_end=2010,␣
↪year=2020)

# Compute RPSS for a specific year
verifier.compute_one_year_rpss(
    obs=obs_data, prob_pred=proba_forecast, clim_year_start=1981, clim_year_end=2010, year=2020
)
```

- **Placeholder Functions**: Some methods (e.g., *taylor_diagram*) are placeholders and require implementation based on specific needs.

- **Gridded Data**: The module currently supports only gridded data validation. Non-gridded validation is not implemented.
- **Performance**: The use of *dask* ensures efficient computation for large datasets, but users should ensure proper chunking of *xarray* DataArrays.
- **Visualization**: Plots are saved to the specified directory and displayed using *matplotlib*. Ensure the output directory exists.

This documentation provides an overview of the Verification module's capabilities, along with example usage for key methods. For detailed information on each method, refer to the source code and docstrings in *was_verification.py*.

1.2.6 Multi-Model Ensemble (MME) Techniques

The *was_mme.py* module provides classes for combining predictions from multiple models, including:

- *WAS_mme_ELM*: Extreme Learning Machine for MME.
- *WAS_mme_EPOELM*: Enhanced Parallel Online Extreme Learning Machine.
- *WAS_mme_MLP*: Multi-Layer Perceptron for MME.
- *WAS_mme_GradientBoosting*: Gradient Boosting for MME.
- *WAS_mme_XGBoosting*: XGBoost for MME.
- *WAS_mme_AdaBoost*: AdaBoost for MME.
- *WAS_mme_LGBM_Boosting*: LightGBM Boosting for MME.
- *WAS_mme_Stack_MLP_RF*: Stacking model with MLP and Random Forest.
- *WAS_mme_Stack_Lasso_RF_MLP*: Stacking model with Lasso, Random Forest, and MLP.
- *WAS_mme_Stack_MLP_Ada_Ridge*: Stacking model with MLP, AdaBoost, and Ridge.
- *WAS_mme_Stack_RF_GB_Ridge*: Stacking model with Random Forest, Gradient Boosting, and Ridge.
- *WAS_mme_Stack_KNN_Tree_SVR*: Stacking model with KNN, Decision Tree, and SVR.
- *WAS_mme_GA*: Genetic Algorithm for MME.

Each MME class includes methods for computing the ensemble model and, where applicable, computing probabilities.

Example Usage with WAS_mme_ELM

```
from wass2s.was_mme import WAS_mme_ELM

# Define ELM parameters
elm_kwargs = {
    'regularization': 10,
    'hidden_layer_size': 4,
    'activation': 'lin',  # Options: 'sigm', 'tanh', 'lin', 'relu'
    'preprocessing': 'none',  # Options: 'minmax', 'std', 'none'
    'n_estimators': 10,
}

# Initialize the MME ELM model
model = WAS_mme_ELM(elm_kwargs=elm_kwargs, dist_method="euclidean")

# Process datasets for MME (user-defined function)
all_model_hdcst, all_model_fcst, obs, best_score = process_datasets_for_mme(
```

(continues on next page)

(continued from previous page)

```
    rainfall.sel(T=slice(str(year_start), str(year_end))),
    gcm=True, ELM_ELR=True, dir_to_save_model="./models",
    best_models=[], scores=[], year_start=1990, year_end=2020,
    model=True, month_of_initialization=3, lead_time=1, year_forecast=2021
)

# Initialize cross-validator
was_mme_gcm = WAS_Cross_Validator(
    n_splits=len(rainfall.sel(T=slice(str(year_start), str(year_end))).get_index("T")),
    nb_omit=2
)

# Perform cross-validation
hindcast_det_gcm, hindcast_prob_gcm = was_mme_gcm.cross_validate(
    model, obs, all_model_hdcst, clim_year_start, clim_year_end
)
```

1.3 wass2s submodules

1.3.1 wass2s.was_download module

class wass2s.was_download.WAS_Download

Bases: object

AgroObsName(*variables={'AGRO.PRCP': ('precipitation_flux', None), 'AGRO.TEMP': ('2m_temperature', '24_hour_mean'), 'AGRO.TMAX': ('2m_temperature', '24_hour_maximum'), 'AGRO.TMIN': ('2m_temperature', '24_hour_minimum')}*)

Generate a dictionary for agro-meteorological observation variables.

Parameters
variables (dict) – Mapping of agro variable short names to full names.

Returns
A dictionary mapping agro variables to their corresponding full names.

Return type
dict

ModelsName(*centre={'BOM_2': 'bom', 'CFSV2_1': 'cfsv2', 'CMC1_1': 'cmc1', 'CMC2_1': 'cmc2', 'CMCC_35': 'cmcc', 'DWD_21': 'dwd', 'DWD_22': 'dwd', 'ECCC_4': 'eccc', 'ECCC_5': 'eccc', 'ECMWF_51': 'ecmwf', 'GFDL_1': 'gfdl', 'JMA_3': 'jma', 'METEOFRANCE_8': 'meteo_france', 'METEOFRANCE_9': 'meteo_france', 'NASA_1': 'nasa', 'NCAR_CCSM4_1': 'ncar_ccsm4', 'NCEP_2': 'ncep', 'NMME_1': 'nmme', 'UKMO_603': 'ukmo', 'UKMO_604': 'ukmo'}*, *variables_1={'DLWR': 'surface_thermal_radiation_downwards', 'DSWR': 'surface_solar_radiation_downwards', 'PRCP': 'total_precipitation', 'SLP': 'mean_sea_level_pressure', 'SST': 'sea_surface_temperature', 'TEMP': '2m_temperature', 'UGRD10': '10m_u_component_of_wind', 'VGRD10': '10m_v_component_of_wind'}*, *variables_2={'HUSS_1000': 'specific_humidity', 'HUSS_850': 'specific_humidity', 'HUSS_925': 'specific_humidity', 'UGRD_1000': 'u_component_of_wind', 'UGRD_850': 'u_component_of_wind', 'UGRD_925': 'u_component_of_wind', 'VGRD_1000': 'v_component_of_wind', 'VGRD_850': 'v_component_of_wind', 'VGRD_925': 'v_component_of_wind'}*)

Generate a combined dictionary of model names and variables. For more information on C3S, browse the

MetaData. For more information on NMME, browse the MetaData.

Parameters

- centre (dict) – Mapping of model identifiers to model names.
- variables_1 (dict) – Mapping of variable short names to full names for category 1.
- variables_2 (dict) – Mapping of variable short names to full names for category 2.

Returns

A combined dictionary with keys as model.variable combinations and values as tuples (model name, variable name).

Return type

dict

ReanalysisName(*centre={'ERA5': 'reanalysis ERA5', 'NOAA': 'NOAA ERSST'}*, *variables_1={'DLWR': 'surface_thermal_radiation_downwards', 'DSWR': 'surface_solar_radiation_downwards', 'PRCP': 'total_precipitation', 'SLP': 'mean_sea_level_pressure', 'SST': 'sea_surface_temperature', 'TEMP': '2m_temperature', 'UGRD10': '10m_u_component_of_wind', 'VGRD10': '10m_v_component_of_wind'}*, *variables_2={'HUSS_1000': 'specific_humidity', 'HUSS_850': 'specific_humidity', 'HUSS_925': 'specific_humidity', 'UGRD_1000': 'u_component_of_wind', 'UGRD_850': 'u_component_of_wind', 'UGRD_925': 'u_component_of_wind', 'VGRD_1000': 'v_component_of_wind', 'VGRD_850': 'v_component_of_wind', 'VGRD_925': 'v_component_of_wind'}*)

Generate a combined dictionary of reanalysis names and variables.

Parameters

- centre (dict) – Mapping of reanalysis identifiers to reanalysis names.
- variables_1 (dict) – Mapping of variable short names to full names for category 1.
- variables_2 (dict) – Mapping of variable short names to full names for category 2.

Returns

A combined dictionary with keys as reanalysis.variable combinations and values as tuples (reanalysis name, variable name).

Return type

dict

WAS_Download_AgroIndicators(*dir_to_save*, *variables*, *year_start*, *year_end*, *area*, *seas=['01', '02', '03']*, *force_download=False*)

Download agro-meteorological indicators for specified variables, years, and months, handling cross-year seasons (e.g., NDJ).

WAS_Download_AgroIndicators_(*dir_to_save*, *variables*, *year_start*, *year_end*, *area*, *seas=['01', '02', '03']*, *force_download=False*)

Download agro-meteorological indicators for specified variables, years, and months.

Parameters

- dir_to_save (str) – Directory to save the downloaded files.
- variables (list) – List of shorthand variables to download (e.g., ["AGRO.PRCP", "AGRO.TMAX"]).
- year_start (int) – Start year for the data to download.
- year_end (int) – End year for the data to download.

- area (list) – Bounding box as [North, West, South, East] for clipping.
- seas (list) – List of month strings representing the season (e.g., ["01", "02", "03"]- for JanFebMar).
- force_download (bool) – If True, forces download even if file exists.

WAS_Download_AgroIndicators_daily(*dir_to_save*, *variables*, *year_start*, *year_end*, *area*, *force_download=False*)

Download daily agro-meteorological indicators for specified variables and years.

Parameters

- dir_to_save (str) – Directory to save the downloaded files.
- variables (list) – List of shorthand variables to download (e.g., ["AGRO.PRCP", "AGRO.TMAX"]).
- year_start (int) – Start year for the data to download.
- year_end (int) – End year for the data to download.
- area (list) – Bounding box as [North, West, South, East] for clipping.
- force_download (bool) – If True, forces download even if file exists.

WAS_Download_CHIRPSv3(*dir_to_save*, *variables*, *year_start*, *year_end*, *area=None*, *season_months=['03', '04', '05']*, *force_download=False*)

Download CHIRPS v3.0 monthly precipitation for a specified cross-year season from year_start to year_end, optionally clipped to 'area', and aggregate them into a single NetCDF file.

WAS_Download_Models(*dir_to_save*, *center_variable*, *month_of_initialization*, *lead_time*, *year_start_hindcast*, *year_end_hindcast*, *area*, *year_forecast=None*, *ensemble_mean=None*, *force_download=False*)

Download seasonal forecast model data for specified center-variable combinations, initialization month, lead times, and years.

Parameters

- dir_to_save (str) – Directory to save the downloaded files.
- center_variable (list) – List of center-variable identifiers (e.g., ["ECMWF_51.PRCP", "UKMO_602.TEMP"]).
- month_of_initialization (int) – Initialization month as an integer (1-12).
- lead_time (list) – List of lead times in months.
- year_start_hindcast (int) – Start year for hindcast data.
- year_end_hindcast (int) – End year for hindcast data.
- area (list) – Bounding box as [North, West, South, East] for clipping.
- year_forecast (int, optional) – Forecast year if downloading forecast data. Defaults to None.
- mean (ensemble) – it's can be median, mean or None. Defaults to None.
- force_download (bool) – If True, forces download even if file exists.

WAS_Download_Models_Daily(*dir_to_save*, *center_variable*, *month_of_initialization*, *day_of_initialization*, *leadtime_hour*, *year_start_hindcast*, *year_end_hindcast*, *area*, *year_forecast=None*, *ensemble_mean=None*, *force_download=False*)

Download daily/sub-daily seasonal forecast model data (original) using 'seasonal-original-single-levels' from the CDS.

Parameters

- dir_to_save (str or Path) – Directory to save the downloaded files.
- center_variable (list) – Each element e.g. "ECMWF_51.PRCP" - left side of '.' is model (ECMWF_51), - right side is variable short code (PRCP).
- month_of_initialization (int) – Initialization month (1-12).
- day_of_initialization (int) – Initialization day (1-31).
- leadtime_hour (list of str) – e.g. ["24", "48", ..., "5160"].
- year_start_hindcast (int) – Start year for hindcast data.
- year_end_hindcast (int) – End year for hindcast data.
- area (list) – Bounding box as [North, West, South, East].
- year_forecast (int, optional) – If provided, downloads that single forecast year. Otherwise downloads hindcast for the specified range.
- ensemble_mean (str, optional) – e.g. "mean", "median", or None.
- force_download (bool) – Force download if True, even if file exists.

WAS_Download_Reanalysis(*dir_to_save*, *center_variable*, *year_start*, *year_end*, *area*, *seas=['01', '02', '03']*, *force_download=False*, *run_avg=3*)

Download reanalysis data for specified center-variable combinations, years, and months, handling cross-year seasons (e.g., NDJ).

WAS_Download_Reanalysis_(*dir_to_save*, *center_variable*, *year_start*, *year_end*, *area*, *seas=['01', '02', '03']*, *force_download=False*)

Download reanalysis data for specified center-variable combinations, years, and months.

Parameters

- dir_to_save (str) – Directory to save the downloaded files.
- center_variable (list) – List of center-variable identifiers (e.g., ["ERA5.PRCP", "MERRA2.TEMP"]).
- year_start (int) – Start year for the data to download.
- year_end (int) – End year for the data to download.
- area (list) – Bounding box as [North, West, South, East] for clipping.
- seas (list) – List of month strings representing the season (e.g., ["01", "02", "03"] for JanFebMar).
- force_download (bool) – If True, forces download even if file exists.

days_in_month(*year*, *month*)

download_nmme_txt_with_progress(*url*, *file_path*, *chunk_size=1024*)

parse_cpt_data_optimized(*file_path*)

wass2s.was_download.plot_map(*extent*, *title='Map'*)

Plots a map with specified geographic extent.

Parameters: - extent: list of float, specifying [west, east, south, north] - title: str, title of the map

1.3.2 wass2s.was_compute_predictand module

class wass2s.was_compute_predictand.WAS_compute_HWSDI

Bases: object

A class to compute the Heat Wave Severity Duration Index (HWSDI), including calculating TXin90 (90th percentile of daily max temperature) and annual counts of heatwave days with at least 6 consecutive hot days.

static calculate_TXin90(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*)

Calculate the daily 90th percentile temperature (TXin90) centered on a 5-day window for each calendar day based on the base period.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.
- base_period_start (str, optional) – Start year of the base period (default is '1961').
- base_period_end (str, optional) – End year of the base period (default is '1990').

Returns

TXin90 for each day of the year.

Return type

xarray.DataArray

compute(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*, *nb_cores=4*)

Compute the Heat Wave Severity Duration Index (HWSDI) for each pixel in a given daily temperature DataArray.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature data, coords = (T, Y, X).
- base_period_start (str, optional) – Start year of the base period for TXin90 calculation (default is '1961').
- base_period_end (str, optional) – End year of the base period for TXin90 calculation (default is '1990').
- nb_cores (int, optional) – Number of parallel processes to use (default is 4).

Returns

HWSDI computed for each pixel.

Return type

xarray.DataArray

count_hot_days(*temperature_data*, *TXin90*)

Count the number of days per year with at least 6 consecutive days where daily maximum temperature is above the 90th percentile.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.

- TXin90 (xarray.DataArray) – 90th percentile temperature for each day of the year.

Returns
Annual count of hot days.

Return type
xarray.DataArray

class wass2s.was_compute_predictand.WAS_compute_HWSDI_Seasonal

Bases: object

A class to compute the Heat Wave Severity Duration Index (HWSDI) for a given season.

static calculate_TXin90(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*, *season=[6, 7, 8]*)

Calculate the daily 90th percentile temperature (TXin90) for each calendar day based on the base period, but only considering the specified season.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.
- base_period_start (str, optional) – Start year of the base period (default is '1961').
- base_period_end (str, optional) – End year of the base period (default is '1990').
- season (list, optional) – List of months to include in the calculation (default is [6, 7, 8] for JJA).

Returns
TXin90 for each day of the selected season.

Return type
xarray.DataArray

compute(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*, *nb_cores=4*, *season=[6, 7, 8]*)

Compute the HWSDI for each pixel in a given daily temperature DataArray for a specific season.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature data, coords = (T, Y, X).
- base_period_start (str, optional) – Start year of the base period for TXin90 calculation (default is '1961').
- base_period_end (str, optional) – End year of the base period for TXin90 calculation (default is '1990').
- nb_cores (int, optional) – Number of parallel processes to use (default is 4).
- season (list, optional) – List of months to include in the calculation (default is [6, 7, 8] for JJA).

Returns
HWSDI computed for each pixel for the given season.

Return type
xarray.DataArray

count_hot_days(*temperature_data*, *TXin90*)

Count the number of days per season with at least 6 consecutive days where daily maximum temperature is above the 90th percentile.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.
- TXin90 (xarray.DataArray) – 90th percentile temperature for each day of the year.

Returns

Seasonal count of hot days.

Return type

xarray.DataArray

class wass2s.was_compute_predictand.WAS_compute_HWSDI_monthly

Bases: object

A class to compute the Heat Wave Severity Duration Index (HWSDI) **monthly**, calculating TXin90 (90th percentile of daily max temperature) and counting heatwave days for each month with at least 6 consecutive hot days.

static calculate_TXin90(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*)

Calculate the monthly 90th percentile temperature (TXin90) centered on a 5-day window for each calendar day based on the base period.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.
- base_period_start (str, optional) – Start year of the base period (default is '1961').
- base_period_end (str, optional) – End year of the base period (default is '1990').

Returns

TXin90 for each month of the year.

Return type

xarray.DataArray

compute(*temperature_data*, *base_period_start='1961'*, *base_period_end='1990'*, *nb_cores=4*)

Compute the Monthly Heat Wave Severity Duration Index (HWSDI) for each pixel in a given daily temperature DataArray.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature data, coords = (T, Y, X).
- base_period_start (str, optional) – Start year of the base period for TXin90 calculation (default is '1961').
- base_period_end (str, optional) – End year of the base period for TXin90 calculation (default is '1990').
- nb_cores (int, optional) – Number of parallel processes to use (default is 4).

Returns

HWSDI computed for each pixel per month.

Return type
xarray.DataArray

count_hot_days(*temperature_data*, *TXin90*)

Count the number of days per month with at least 6 consecutive days where daily maximum temperature is above the 90th percentile.

Parameters

- temperature_data (xarray.DataArray) – Daily maximum temperature with time dimension.
- TXin90 (xarray.DataArray) – 90th percentile temperature for each month.

Returns
Monthly count of hot days.

Return type
xarray.DataArray

class wass2s.was_compute_predictand.WAS_compute_cessation(*user_criteria=None*)

Bases: object

A class to compute cessation dates based on soil moisture balance for different regions and criteria, leveraging parallel computation for efficiency.

static adjust_duplicates(*series*, *increment=1e-05*)

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

cessation_function(*x*, *ijour_dem_cal*, *idebut*, *ETP*, *Cap_ret_maxi*, *irch_fin*)

Compute cessation date using soil moisture balance criteria.

compute(*daily_data*, *nb_cores*)

Compute cessation dates for each pixel using criteria based on regions.

compute_insitu(*daily_df*)

static day_of_year(*i*, *dem_rech1*)

Given a year ‘i’ and a month-day string ‘dem_rech1’ (e.g., ‘07-23’), return the 1-based day of the year.

default_criteria = {0: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'date_dry_soil': '01-01', 'end_search': '09-30', 'start_search': '09-01', 'zone_name': 'Sahel100_0mm'}, 1: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'date_dry_soil': '01-01', 'end_search': '10-05', 'start_search': '09-01', 'zone_name': 'Sahel200_100mm'}, 2: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'date_dry_soil': '01-01', 'end_search': '11-10', 'start_search': '09-01', 'zone_name': 'Sahel400_200mm'}, 3: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'date_dry_soil': '01-01', 'end_search': '11-15', 'start_search': '09-15', 'zone_name': 'Sahel600_400mm'}, 4: {'Cap_ret_maxi': 70, 'ETP': 4.5, 'date_dry_soil': '01-01', 'end_search': '11-30', 'start_search': '10-01', 'zone_name': 'Soudan'}, 5: {'Cap_ret_maxi': 70, 'ETP': 4.0, 'date_dry_soil': '01-01', 'end_search': '12-01', 'start_search': '10-15', 'zone_name': 'Golfe_Of_Guinea'}}

rainf_zone(*daily_data*)

static transform_cdt(*df*)

Transform a DataFrame with:

- Row 0 = LON
- Row 1 = LAT

- Row 2 = ELEV
- Rows 3+ = daily data (or any date) with 'ID' column containing dates.

Returns an xarray DataArray with coords = (T, Y, X), variable = 'Observation'.

class wass2s.was_compute_predictand.WAS_compute_cessation_dry_spell(*user_criteria=None*)

Bases: object

A class for computing the longest dry spell length after the onset of a rainy season, based on user-defined criteria.

static adjust_duplicates(*series*, *increment=1e-05*)

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

compute(*daily_data*, *nb_cores*)

Compute the longest dry spell length after the rainy season onset for each pixel in the given daily rainfall DataArray, using different criteria (both for onset and cessation) based on isohyet zones.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).
- nb_cores (int) – Number of parallel processes (workers) to use.

Returns

Array with the longest dry spell length per pixel.

Return type

xarray.DataArray

compute_insitu(*daily_df*)

static day_of_year(*i*, *dem_rech1*)

Convert year i and MM-DD string dem_rech1 (e.g., '07-23') into a 1-based day of the year.

default_criteria = {0: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'cumulative': 10, 'date_dry_soil': '01-01', 'end_search1': '08-15', 'end_search2': '09-30', 'nbjour': 40, 'number_dry_days': 25, 'start_search1': '05-01', 'start_search2': '09-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel100_0mm'}, 1: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'cumulative': 15, 'date_dry_soil': '01-01', 'end_search1': '08-15', 'end_search2': '10-05', 'nbjour': 40, 'number_dry_days': 25, 'start_search1': '05-15', 'start_search2': '09-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel200_100mm'}, 2: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'cumulative': 15, 'date_dry_soil': '01-01', 'end_search1': '07-31', 'end_search2': '11-10', 'nbjour': 40, 'number_dry_days': 20, 'start_search1': '05-01', 'start_search2': '09-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel400_200mm'}, 3: {'Cap_ret_maxi': 70, 'ETP': 5.0, 'cumulative': 20, 'date_dry_soil': '01-01', 'end_search1': '07-31', 'end_search2': '11-15', 'nbjour': 45, 'number_dry_days': 20, 'start_search1': '03-15', 'start_search2': '09-15', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel600_400mm'}, 4: {'Cap_ret_maxi': 70, 'ETP': 4.5, 'cumulative': 20, 'date_dry_soil': '01-01', 'end_search1': '07-31', 'end_search2': '11-30', 'nbjour': 50, 'number_dry_days': 10, 'start_search1': '03-15', 'start_search2': '10-01', 'thrd_rain_day': 0.85, 'zone_name': 'Soudan'}, 5: {'Cap_ret_maxi': 70, 'ETP': 4.0, 'cumulative': 20, 'date_dry_soil': '01-01', 'end_search1': '06-15', 'end_search2': '12-01', 'nbjour': 50, 'number_dry_days': 10, 'start_search1': '02-01', 'start_search2': '10-15', 'thrd_rain_day': 0.85, 'zone_name': 'Golfe_Of_Guinea'}}

dry_spell_cessation_function(*x*, *idebut1*, *cumul*, *nbsec*, *jour_pluvieux*, *irch_fin1*, *idebut2*, *ijour_dem_cal*, *ETP*, *Cap_ret_maxi*, *irch_fin2*, *nbjour*)

Computes the longest dry spell length after the onset and determines the cessation date (when soil water returns to 0) based on water balance, then checks for a dry spell.

Parameters

- x (array-like) – Daily rainfall or similar values.
- idebut1 (int) – Start index to begin searching for the onset.
- cumul (float) – Cumulative rainfall threshold to trigger onset.
- nbsec (int) – Maximum number of dry days allowed in the sequence.
- jour_pluvieux (float) – Minimum rainfall to consider a day as rainy.
- irch_fin1 (int) – Maximum index limit for the onset search.
- idebut2 (int) – Start index for the cessation search.
- ijour_dem_cal (int) – Start index from which the water balance is calculated.
- ETP (float) – Daily evapotranspiration (mm).
- Cap_ret_maxi (float) – Maximum soil water retention capacity (mm).
- irch_fin2 (int) – Maximum index limit for the cessation search.
- nbjour (int) – Number of days after onset to check for the dry spell.

Returns

Length of the longest dry spell sequence after onset and before soil water returns to zero, or NaN if not found.

Return type

float

rainf_zone(*daily_data*)

static transform_cdt(*df*)

Transform a DataFrame with:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data (or any date) with 'ID' column containing dates.

Returns an xarray DataArray with coords = (T, Y, X), variable = 'Observation'.

class wass2s.was_compute_predictand.WAS_compute_onset(*user_criteria=None*)

Bases: object

A class that encapsulates methods for transforming precipitation data from different formats (CPT, CDT) and computing onset dates based on rainfall criteria.

static adjust_duplicates(*series*, *increment=1e-05*)

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

compute(*daily_data*, *nb_cores*)

Compute onset dates for each pixel in a given daily rainfall DataArray using different criteria based on isohyet zones.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).

- nb_cores (int) – Number of parallel processes to use.

Returns
Array with onset dates computed per pixel.

Return type
xarray.DataArray

compute_insitu(*daily_df*)

static day_of_year(*i*, *dem_rech1*)

Given a year 'i' and a month-day string 'dem_rech1' (e.g., '07-23'), return the day of the year (1-based).

default_criteria = {0: {'cumulative': 10, 'end_search': '08-30', 'number_dry_days': 25, 'start_search': '06-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel100_0mm'}, 1: {'cumulative': 15, 'end_search': '08-15', 'number_dry_days': 25, 'start_search': '05-15', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel200_100mm'}, 2: {'cumulative': 15, 'end_search': '07-31', 'number_dry_days': 20, 'start_search': '05-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel400_200mm'}, 3: {'cumulative': 20, 'end_search': '07-31', 'number_dry_days': 20, 'start_search': '03-15', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel600_400mm'}, 4: {'cumulative': 20, 'end_search': '07-31', 'number_dry_days': 10, 'start_search': '03-15', 'thrd_rain_day': 0.85, 'zone_name': 'Soudan'}, 5: {'cumulative': 20, 'end_search': '06-15', 'number_dry_days': 10, 'start_search': '02-01', 'thrd_rain_day': 0.85, 'zone_name': 'Golfe_Of_Guinea'}}

onset_function(*x*, *idebut*, *cumul*, *nbsec*, *jour_pluvieux*, *irch_fin*)

Calculate the onset date of a season based on cumulative rainfall criteria.

Parameters

- x (array-like) – Daily rainfall or similar values.
- idebut (int) – Start index to begin searching for the onset.
- cumul (float) – Cumulative rainfall threshold to trigger onset.
- nbsec (int) – Maximum number of dry days allowed in the sequence.
- jour_pluvieux (float) – Minimum rainfall to consider a day as rainy.
- irch_fin (int) – Maximum index limit for the onset.

Returns
Index of the onset date or NaN if onset not found.

Return type
int or float

rainf_zone(*daily_data*)

static transform_cdt(*df*)

Transform a DataFrame with:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data (or any date) with 'ID' column containing dates.

Returns an xarray DataArray with coords = (T, Y, X), variable = 'Observation'.

class wass2s.was_compute_predictand.WAS_compute_onset_dry_spell(*user_criteria=None*)

Bases: object

A class for computing the longest dry spell length after the onset of a rainy season, based on user-defined criteria.

static adjust_duplicates(*series*, *increment=1e-05*)

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

compute(*daily_data*, *nb_cores*)

Compute the longest dry spell length after the onset for each pixel in a given daily rainfall DataArray, using different criteria based on isohyet zones.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).
- nb_cores (int) – Number of parallel processes to use.

Returns

Array with the longest dry spell length per pixel.

Return type

xarray.DataArray

compute_insitu(*daily_df*)

static day_of_year(*i*, *dem_rech1*)

Given a year ‘i’ and a month-day string ‘dem_rech1’ (e.g., ‘07-23’), return the 1-based day of the year.

default_criteria = {0: {'cumulative': 10, 'end_search': '08-30', 'nbjour': 40, 'number_dry_days': 25, 'start_search': '06-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel100_0mm'}, 1: {'cumulative': 15, 'end_search': '08-15', 'nbjour': 40, 'number_dry_days': 25, 'start_search': '05-15', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel200_100mm'}, 2: {'cumulative': 15, 'end_search': '07-31', 'nbjour': 40, 'number_dry_days': 20, 'start_search': '05-01', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel400_200mm'}, 3: {'cumulative': 20, 'end_search': '07-31', 'nbjour': 45, 'number_dry_days': 20, 'start_search': '03-15', 'thrd_rain_day': 0.85, 'zone_name': 'Sahel600_400mm'}, 4: {'cumulative': 20, 'end_search': '07-31', 'nbjour': 50, 'number_dry_days': 10, 'start_search': '03-15', 'thrd_rain_day': 0.85, 'zone_name': 'Soudan'}, 5: {'cumulative': 20, 'end_search': '06-15', 'nbjour': 50, 'number_dry_days': 10, 'start_search': '02-01', 'thrd_rain_day': 0.85, 'zone_name': 'Golfe_Of_Guinea'}}

dry_spell_onset_function(*x*, *idebut*, *cumul*, *nbsec*, *jour_pluvieux*, *irch_fin*, *nbjour*)

Calculate the onset date of a season based on cumulative rainfall criteria, and determine the longest dry spell sequence within a specified period after the onset.

dry_spell_onset_function_(*x*, *idebut*, *cumul*, *nbsec*, *jour_pluvieux*, *irch_fin*, *nbjour*)

Calculate the onset date of a season based on cumulative rainfall criteria, and determine the longest dry spell sequence within a specified period after the onset.

Parameters

- x (array-like) – Daily rainfall or similar values.
- idebut (int) – Start index to begin searching for the onset.
- cumul (float) – Cumulative rainfall threshold to trigger onset.
- nbsec (int) – Maximum number of dry days allowed in the sequence.
- jour_pluvieux (float) – Minimum rainfall to consider a day as rainy.

- irch_fin (int) – Maximum index limit for the onset.
- nbjour (int) – Number of days to check for the longest dry spell after onset.

Returns
Length of the longest dry spell sequence after onset or NaN if onset not found.

Return type
float

rainf_zone(*daily_data*)

static transform_cdt(*df*)

Transform a DataFrame with:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data (or any date) with 'ID' column containing dates.

Returns an xarray DataArray with coords = (T, Y, X), variable = 'Observation'.

class wass2s.was_compute_predictand.WAS_count_dry_spells

Bases: object

A class to compute the number of dry spells within a specified period (onset to cessation) for each pixel or station in a daily rainfall dataset.

static adjust_duplicates(*series*, *increment=1e-05*)

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

compute(*daily_data*, *onset_date*, *cessation_date*, *dry_spell_length*, *dry_threshold*, *nb_cores*)

Compute the number of dry spells for each pixel within the onset and cessation period in a daily xarray DataArray.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).
- onset_date (xarray.DataArray) – DataArray containing onset dates for each pixel.
- cessation_date (xarray.DataArray) – DataArray containing cessation dates for each pixel.
- dry_spell_length (int) – The length of a dry spell to count.
- dry_threshold (float) – Rainfall threshold to classify a day as "dry."
- nb_cores (int) – Number of parallel processes to use.

Returns
An array with the count of dry spells per pixel.

Return type
xarray.DataArray

compute_insitu(*daily_df*, *onset_df_cpt*, *cessation_df_cpt*, *dry_spell_length*, *dry_threshold=1.0*)

Compute the number of dry spells (of length = dry_spell_length) between the onset and cessation dates for in-situ stations (CDT format).

Returns a DataFrame in CPT format:

- Row 0: ["LAT", lat_stn1, lat_stn2, ...]
- Row 1: ["LON", lon_stn1, lon_stn2, ...]
- Subsequent rows: [year, station1_value, station2_value, ...]

Parameters

- daily_df (pd.DataFrame) – CDT rainfall data (ID column = date, station columns).
- onset_df_cpt (pd.DataFrame) – CPT-format DataFrame containing onset dates (as returned by some method).
- cessation_df_cpt (pd.DataFrame) – CPT-format DataFrame containing cessation dates.
- dry_spell_length (int) – The length of the dry spell to look for.
- dry_threshold (float, optional) – Rainfall threshold below which a day is considered "dry." Defaults to 1.0 mm.

Returns
Final dry-spell counts in CPT pivot format.

Return type
pd.DataFrame

static count_dry_spells(*x*, *onset*, *cessation*, *dry_spell_length*, *dry_threshold*)

Count the number of dry spells of a specific length between onset and cessation dates.

Parameters

- x (array-like) – Daily rainfall values.
- onset (int) – Start index for the calculation (onset date).
- cessation (int) – End index for the calculation (cessation date).
- dry_spell_length (int) – The length of a dry spell to count.
- dry_threshold (float) – Rainfall threshold to classify a day as "dry."

Returns
The number of dry spells of the specified length (NaN if invalid).

Return type
int or float

static transform_cdt(*df*)

Transform a DataFrame with:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data with 'ID' column containing dates.

Returns a DataFrame with columns like:
DATE | STATION | VALUE | LON | LAT | ELEV | MEAN_ANNUAL_RAINFALL | zonename

static transform_cpt(*df*, *missing_value=None*)

Transform a DataFrame in CPT format with:

- Row 0 = LAT

- Row 1 = LON
- Rows 2+ = numeric year data in wide format (stations in columns).

Returns a DataFrame with columns like:
YEAR | STATION | VALUE | LAT | LON

class wass2s.was_compute_predictand.WAS_count_rainy_days

Bases: object

A class to compute the number of rainy days between onset and cessation dates for each pixel or station in a daily rainfall dataset.

compute(*daily_data*, *onset_date*, *cessation_date*, *rain_threshold*, *nb_cores*)

Compute the number of rainy days for each pixel between onset and cessation dates.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).
- onset_date (xarray.DataArray) – DataArray containing onset dates for each pixel.
- cessation_date (xarray.DataArray) – DataArray containing cessation dates for each pixel.
- rain_threshold (float) – Rainfall threshold to classify a day as "rainy."
- nb_cores (int) – Number of parallel processes to use.

Returns
Array with the count of rainy days per pixel.

Return type
xarray.DataArray

compute_insitu(*daily_df*, *onset_df_cpt*, *cessation_df_cpt*, *rain_threshold=0.85*)

Compute, for in-situ stations (CDT data), the number of rainy days between onset and cessation, for each station and year.

Parameters

- daily_df (pd.DataFrame) – CDT precipitation data (ID column = date; columns = stations). Follows the standard CDT format.
- onset_df_cpt (pd.DataFrame) – Result of *WAS_compute_onset.compute_insitu(...)* for onset (CPT format).
- cessation_df_cpt (pd.DataFrame) – Same format for cessation (CPT format).
- rain_threshold (float, optional) – Precipitation threshold for counting a day as "rainy," by default 0.85 mm.

Returns
df_final – The count of rainy days in CPT pivot format.

Return type
pd.DataFrame

static count_rainy_days(*x*, *onset_date*, *cessation_date*, *rain_threshold*)

Count the number of rainy days between onset and cessation dates.

Parameters

- x (array-like) – Daily rainfall values.

- onset_date (int) – Start index for the calculation (onset date).
- cessation_date (int) – End index for the calculation (cessation date).
- rain_threshold (float) – Rainfall threshold to classify a day as "rainy."

Returns
Number of rainy days (returns NaN if data is invalid).

Return type
int or float

static transform_cdt(*df*)

Transform a DataFrame in CDT format into a standardized long DataFrame.

CDT format assumptions:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data with 'ID' column holding dates in YYYYMMDD format.

This method returns a DataFrame with columns:
DATE, STATION, VALUE, LON, LAT, ELEV, (optional) MEAN_ANNUAL_RAINFALL, zonename

class wass2s.was_compute_predictand.WAS_count_wet_spells

Bases: object

A class to compute the number of wet spells within a specified period (onset to cessation) for each pixel or station in a daily rainfall dataset.

compute(*daily_data*, *onset_date*, *cessation_date*, *wet_spell_length*, *wet_threshold*, *nb_cores*)

Compute the number of wet spells for each pixel within the onset and cessation period in a daily xarray DataArray.

Parameters

- daily_data (xarray.DataArray) – Daily rainfall data, coords = (T, Y, X).
- onset_date (xarray.DataArray) – DataArray containing onset dates for each pixel.
- cessation_date (xarray.DataArray) – DataArray containing cessation dates for each pixel.
- wet_spell_length (int) – The length of a wet spell to count.
- wet_threshold (float) – Rainfall threshold to classify a day as "wet."
- nb_cores (int) – Number of parallel processes to use.

Returns
Array with the count of wet spells per pixel.

Return type
xarray.DataArray

compute_insitu(*daily_df*, *onset_df_cpt*, *cessation_df_cpt*, *wet_spell_length*, *wet_threshold=1.0*)

Compute the number of wet spells (of length = wet_spell_length) between onset and cessation for in-situ stations (CDT data).

Returns a DataFrame in CPT format:

- Row 0: ["LAT", lat_station1, lat_station2, ...]

- Row 1: ["LON", lon_station1, lon_station2, ...]
- Then one row per year: [year, station1_value, station2_value, ...]

Parameters

- daily_df (pd.DataFrame) – CDT rainfall data (ID column = date, station columns).
- onset_df_cpt (pd.DataFrame) – CPT-format DataFrame with onset dates (same station columns).
- cessation_df_cpt (pd.DataFrame) – CPT-format DataFrame with cessation dates (same station columns).
- wet_spell_length (int) – The length of a wet spell to count.
- wet_threshold (float, optional) – Rainfall threshold classifying a day as "wet." Defaults to 1.0 mm.

Returns
Final wet-spell counts in CPT pivot format.

Return type
pd.DataFrame

static count_wet_spells(*x*, *onset_date*, *cessation_date*, *wet_spell_length*, *wet_threshold*)

Count the number of wet spells of a specific length between onset and cessation dates.

Parameters

- x (array-like) – Daily rainfall values.
- onset_date (int) – Start index for the calculation (onset date).
- cessation_date (int) – End index for the calculation (cessation date).
- wet_spell_length (int) – The length of a wet spell to count.
- wet_threshold (float) – Rainfall threshold to classify a day as "wet."

Returns
The number of wet spells of the specified length (NaN if data is invalid).

Return type
int or float

static transform_cdt(*df*)

Transform a CDT-format DataFrame into a standard table.

CDT format assumptions:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data, 'ID' column has dates in YYYYMMDD.

Returns a DataFrame with columns:
[DATE, STATION, VALUE, LON, LAT, ELEV, MEAN_ANNUAL_RAINFALL, zonename]

class wass2s.was_compute_predictand.WAS_r95_99p(*base_period: slice*, *season: list = None*)

Bases: object

A class to compute the R95p and R99p climate indices using either: - Dask-enabled xarray for large raster/time-series - An "insitu" method for station-based (CDT) data.

compute_insitu_r95p(*df_cdt: DataFrame*) → DataFrame

Compute R95p index (total precipitation on days above the daily 95th percentile) for station-based data in CDT format.

Parameters
df_cdt (pd.DataFrame) – CDT-format DataFrame (rows 0..2 = LON/LAT/ELEV, row 3+ = daily data).

Returns
df_final – A DataFrame in CPT format with the R95p values pivoted by station vs. year.

Return type
pd.DataFrame

compute_insitu_r99p(*df_cdt: DataFrame*) → DataFrame

Compute R99p index (total precipitation on days above the daily 99th percentile) for station-based data in CDT format.

Parameters
df_cdt (pd.DataFrame) – CDT-format DataFrame.

Returns
df_final

Return type
pd.DataFrame (CPT format)

compute_r95p(*pr: DataArray*) → DataArray

Existing method for xarray-based data (unchanged).

compute_r99p(*pr: DataArray*) → DataArray

Existing method for xarray-based data (unchanged).

static transform_cdt(*df*)

Transform a DataFrame in CDT format into a standardized long DataFrame.

CDT format assumptions:

- Row 0 = LON
- Row 1 = LAT
- Row 2 = ELEV
- Rows 3+ = daily data with 'ID' column holding dates in YYYYMMDD format.

Returns a DataFrame with columns:
[DATE, STATION, VALUE, LON, LAT, ELEV]

1.3.3 wass2s.was_merge_predictand module

class wass2s.was_merge_predictand.WAS_Merging(*df: DataFrame*, *da: Dataset*, *date_month_day: str = '08-01'*)

Bases: object

adjust_duplicates(*series: Series*, *increment: float = 1e-05*) → Series

If any values in the Series repeat, nudge them by a tiny increment so that all are unique (to avoid indexing collisions).

auto_select_kriging_parameters(*df: DataFrame*, *x_col: str = 'X'*, *y_col: str = 'Y'*, *z_col: str = 'residuals'*, *variogram_models: list = None*, *nlags_range=range(3, 10)*, *n_splits: int = 5*, *random_state: int = 42*, *verbose: bool = False*, *enable_plotting: bool = False*)

Automatically selects the best variogram_model and nlags for Ordinary Kriging using cross-validation.

Returns

- best_model (str)
- best_nlags (int)
- ok_best (OrdinaryKriging object)
- results_df (pd.DataFrame)

multiplicative_bias(*missing_value: float = -999.0*, *do_cross_validation: bool = False) -> (<class 'xarray.core.dataarray.DataArray'>*, *<class 'pandas.core.frame.DataFrame'>*)

Apply multiplicative bias correction to gridded predictions using ground observations.

This method performs bias adjustment for each time step by comparing standardized observations and model estimates, interpolating residuals using kriging, and applying a multiplicative correction across the spatial domain.

Parameters

- missing_value (float, optional) – Value used to represent missing data in the input observational CSV, by default -999.0.
- do_cross_validation (bool, optional) – If True, perform Leave-One-Out Cross-Validation to estimate kriging performance, by default False.

Returns

- *xr.DataArray* – The bias-corrected spatial dataset over time (with dimensions: T, Y, X), masked over valid areas.
- *pd.DataFrame or None* – DataFrame containing LOOCV RMSE per time step, or None if cross-validation was not performed.

neural_network_kriging(*missing_value: float = -999.0*, *do_cross_validation: bool = False) -> (<class 'xarray.core.dataarray.DataArray'>*, *pandas.core.frame.DataFrame | None*)

Performs a two-step spatial bias adjustment for each time slice, leveraging:

1) **Neural Network** (MLPRegressor) to capture non-linear relationships between the large-scale estimations (predictors) and in-situ observations (predictands).
2) **Kriging** of the residuals (observation - NN_prediction) to spatially interpolate the remaining error structure under the assumption that these residuals are stationary and exhibit spatial autocorrelation.

Parameters

- missing_value (float, optional) – Value used to fill missing data in the input station dataset, by default -999.0
- do_cross_validation (bool, optional) – Whether or not to perform cross-validation (e.g., leave-one-out or k-fold) during kriging parameter selection, by default False.

Returns

- *xr.DataArray* – Bias-adjusted field over the domain, with the same spatial dimensions as the original input (and time dimension if applicable).
- *pd.DataFrame or None* – If *do_cross_validation=True*, returns a DataFrame with RMSE records from CV; otherwise, None.

Notes

- The hyperparameter search for the MLP uses GridSearchCV with MSE-based scoring.
- Kriging assumes the residuals are reasonably stationary and spatially correlated.

neural_network_kriging_(*missing_value: float = -999.0*, *do_cross_validation: bool = False) -> (<class 'xarray.core.dataarray.DataArray'>*, *<class 'pandas.core.frame.DataFrame'>*)

Performs spatial bias adjustment for each time slice using:

1) Neural Network
2) Kriging of residuals

Optionally performs LOO cross-validation for each time.

plot_merging_comparaison(*df_Obs*, *da_estimated*, *da_corrected*, *missing_value=-999.0*)

regression_kriging(*missing_value: float = -999.0*, *do_cross_validation: bool = False) -> (<class 'xarray.core.dataarray.DataArray'>*, *<class 'pandas.core.frame.DataFrame'>*)

Performs spatial bias adjustment for each time slice using:

1) Linear Regression
2) Kriging of residuals

Optionally performs LOO cross-validation for each time.

simple_bias_adjustment(*missing_value: float = -999.0*, *do_cross_validation: bool = False) -> (<class 'xarray.core.dataarray.DataArray'>*, *<class 'pandas.core.frame.DataFrame'>*)

Performs spatial bias adjustment for each time slice. Optionally does leave-one-out (LOO) cross-validation to compute RMSE at each time.

Returns

- *xr.DataArray* – Concatenated bias-adjusted values along the time dimension.
- *pd.DataFrame or None* – A DataFrame containing the LOO RMSE for each time if do_cross_validation=True. Otherwise, returns None.

transform_cdt(*df: DataFrame*) → DataArray

transform_cpt(*df: DataFrame*, *missing_value: float = -999.0*) → DataArray

1.3.4 wass2s.was_cross_validate module

class wass2s.was_cross_validate.CustomTimeSeriesSplit(*n_splits*)

Bases: object

Custom time series cross-validator for splitting data into training and test sets.

Ensures temporal ordering is maintained by generating training and test indices suitable for time series data, with an option to omit samples after the test index.

Parameters
n_splits (int) – Number of splits for the cross-validation.

get_n_splits(*X=None*, *y=None*, *groups=None*)

Return the number of splits for the cross-validation.

Parameters

- X (array-like, optional) – The data to be split (ignored in this implementation).
- y (array-like, optional) – The target variable (ignored in this implementation).
- groups (array-like, optional) – Group labels for the samples (ignored in this implementation).

Returns
The number of splits configured for the cross-validator.

Return type
int

split(*X*, *nb_omit*, *y=None*, *groups=None*)

Generate indices to split data into training and test sets.

Yields training indices before the test index (excluding a specified number of samples after the test index) and test indices for each split.

Parameters

- X (array-like) – The data to be split, typically time series data.
- nb_omit (int) – Number of samples to omit from training after the test index to avoid data leakage.
- y (array-like, optional) – The target variable (ignored in this implementation).
- groups (array-like, optional) – Group labels for the samples (ignored in this implementation).

Yields

- **train_indices** (*ndarray*) – The training set indices for the current split.
- **test_indices** (*list*) – The test set indices for the current split.

class wass2s.was_cross_validate.WAS_Cross_Validator(*n_splits*, *nb_omit*)

Bases: object

Performs cross-validation for time series forecasting models using a custom time series split.

This class wraps a custom time series cross-validator to evaluate forecasting models, handling both deterministic hindcasts and probabilistic (tercile) predictions.

Parameters

- n_splits (int) – Number of splits for the cross-validation.
- nb_omit (int) – Number of samples to omit from training after the test index to prevent data leakage.

cross_validate(*model*, *Predictant*, *Predictor=None*, *clim_year_start=None*, *clim_year_end=None*, ***model_params*)

Perform cross-validation to compute deterministic hindcasts and tercile probabilities.

Iterates over time series splits, trains the model on training data, and generates predictions for test data. Supports special handling for specific model types (e.g., CCA, Analog, ELM, ELR, and various machine learning models).

Parameters

- model (object) – The forecasting model instance to evaluate.
- Predictant (xarray.DataArray) – Target dataset with dimensions ('T', 'Y', 'X') or ('T', 'M', 'Y', 'X').
- Predictor (xarray.DataArray, optional) – Predictor dataset with dimensions ('T', 'M', 'Y', 'X') or ('T', 'features'). Required for most models except specific cases like WAS_Analog.
- clim_year_start (int or str, optional) – Start year of the climatology period for standardization and probability calculations.
- clim_year_end (int or str, optional) – End year of the climatology period for standardization and probability calculations.
- **model_params (dict) – Additional keyword arguments to pass to the model's compute_model method.

Returns

If the model supports probability calculations (has compute_prob method):

•hindcast_det
[xarray.DataArray] Deterministic hindcast results with dimensions ('T', 'Y', 'X') or ('probability', 'T', 'Y', 'X') for specific models.

•hindcast_prob
[xarray.DataArray] Tercile probabilities with dimensions ('probability', 'T', 'Y', 'X'), where 'probability' includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

If the model does not support probability calculations (e.g., WAS_mme_ELR):

•hindcast_det
[xarray.DataArray] Deterministic hindcast results with dimensions ('T', 'Y', 'X').

Return type
tuple or xarray.DataArray

get_model_params(*model*)

Retrieve parameters required for the model's compute_model method.

Extracts parameters dynamically from the model's attributes that match the argument names of its compute_model method.

Parameters
model (object) – The forecasting model instance to inspect.

Returns
Dictionary of parameter names and values to pass to the model's compute_model method.

Return type
dict

1.3.5 wass2s.was_linear_models module

class wass2s.was_linear_models.WAS_ElasticNet_Model(*alpha_range=None*, *l1_ratio_range=None*, *n_clusters=5*, *nb_cores=1*, *dist_method='gamma'*)

Bases: object

A class to implement the ElasticNet model for spatiotemporal regression with clustering, cross-validation, and probabilistic predictions.

Attributes:

alpha_range
[numpy.ndarray] Range of alpha values (regularization strength) to explore. Default is from 10^-6 to 10^2.

l1_ratio_range
[list] Range of l1 ratio values (mixing between L1 and L2 penalties). Default is [0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1].

n_clusters
[int] Number of clusters for KMeans-based spatial grouping. Default is 5.

nb_cores
[int] Number of CPU cores for parallel computations. Default is 1.

dist_method
[str] Distribution method for tercile probability calculations (e.g. 'gamma','t','normal','lognormal','nonparam'). Default = "gamma".

Methods:

fit_predict(x, y, x_test, y_test=None, alpha=None, l1_ratio=None)
Fits an ElasticNet model to training data and predicts values for test data.

compute_hyperparameters(predictand, predictor)
Computes the optimal alpha and l1 ratio for each cluster using cross-validation.

compute_model(X_train, y_train, X_test, y_test, alpha, l1_ratio)
Performs parallelized ElasticNet modeling for spatiotemporal data.

calculate_tercile_probabilities(...)
Various static methods for computing tercile probabilities (t, gamma, normal, etc.).

compute_prob(Predictant, clim_year_start, clim_year_end, Predictor, hindcast_det)
Computes probabilistic hindcasts for tercile categories based on climatological terciles.

forecast(Predictant, clim_year_start, clim_year_end, Predictor, hindcast_det, Predictor_for_year, alpha, l1_ratio)
Generates a single-year forecast using ElasticNet with alpha + l1_ratio, and computes tercile probabilities based on the chosen method.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method for tercile probabilities.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Computes the optimal alpha and l1 ratio for each cluster using cross-validation.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*, *l1_ratio*)

Performs parallelized ElasticNet modeling for spatiotemporal data. Returns [error, prediction] per grid cell.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns

hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type

xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test=None*, *alpha=None*, *l1_ratio=None*)

Fits an ElasticNet model to training data and predicts on test data.

Parameters

- x (ndarray, shape (n_samples, n_features))
- y (ndarray, shape (n_samples,))
- x_test (ndarray, shape (n_features,) or (1, n_features))
- y_test (float or None)
- alpha (float)
- l1_ratio (float)

Returns

[error, prediction] if y_test is provided; [prediction] if y_test is None.

Return type

np.ndarray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*, *l1_ratio*)

Generates a single-year forecast using ElasticNet with (alpha, l1_ratio), and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed data (T, Y, X).
- clim_year_start (int) – Climatology reference period.

- clim_year_end (int) – Climatology reference period.
- Predictor (xarray.DataArray) – Historical predictor data (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=2, T, Y, X).
- Predictor_for_year (xarray.DataArray) – Single-year predictor data (features,).
- alpha (xarray.DataArray) – Spatial map of alpha values (Y, X).
- l1_ratio (xarray.DataArray) – Spatial map of l1_ratio values (Y, X).

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, forecast]. For a real forecast scenario, 'error' is NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA].

class wass2s.was_linear_models.WAS_LassoLars_Model(*alpha_range=None*, *n_clusters=5*, *nb_cores=1*, *dist_method='gamma'*)

Bases: object

A class to implement the Lasso Least Angle Regression (LassoLars) model for spatiotemporal climate prediction.

This model is designed to work with climate data by clustering spatial regions, computing hyperparameters for each cluster, and fitting a LassoLars model for predictions. The model is optimized for parallel execution using Dask.

Parameters

- alpha_range (array-like, optional) – Range of alpha values for the LassoLars model. Default is np.array([10**i for i in range(-6, 6)]).
- n_clusters (int, default=5) – Number of clusters to partition the spatial data.
- nb_cores (int, default=1) – Number of cores for parallel processing.
- dist_method (str, default='gamma') – Distribution method for calculating tercile probabilities. One of {'gamma','t','normal','lognormal','nonparam'}.

fit_predict(*x*, *y*, *x_test*, *y_test*, *alpha*)

Fits the LassoLars model to the training data and predicts values for the test data.

compute_hyperparameters(*predictand*, *predictor*)

Computes cluster-wise optimal alpha values for LassoLars using cross-validation.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*)

Fits and predicts the LassoLars model using Dask for parallel execution.

calculate_tercile_probabilities(...)

Calculates tercile probabilities using the chosen distribution method (Student's t, gamma, etc.).

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*)

Computes the tercile probabilities for hindcast predictions using a climatological period.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*)

Generates a forecast for a single time step and computes tercile probabilities.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Computes cluster-wise optimal alpha values for LassoLars using cross-validation.

Parameters

- predictand (xarray.DataArray) – The response variable for clustering and training (dims: T, Y, X).
- predictor (array-like) – Predictor variables used for fitting the model (dims: T, features).

Returns

- **alpha_array** (*xarray.DataArray*) – Cluster-wise optimal alpha values.
- **Cluster** (*xarray.DataArray*) – Cluster assignment for each spatial point.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*)

Fits and predicts the LassoLars model using Dask for parallel execution.

Parameters

- X_train (xarray.DataArray) – Training predictor data (dims: T, features).
- y_train (xarray.DataArray) – Training response variable (dims: T, Y, X).
- X_test (xarray.DataArray) – Test predictor data (dims: features,) or broadcastable to (Y, X).
- y_test (xarray.DataArray) – Test response variable (dims: Y, X).
- alpha (xarray.DataArray) – Cluster-wise optimal alpha values (dims: Y, X).

Returns

dims (output=2, Y, X) => [error, prediction].

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*, *alpha*)

Fits the LassoLars model to the training data and predicts values for the test data.

Parameters

- x (array-like) – Training predictors (shape: [n_samples, n_features]).
- y (array-like) – Training response variable (shape: [n_samples,]).
- x_test (array-like) – Test predictors (shape: [n_features,] or [1, n_features]).
- y_test (float) – Test response (scalar or shape [1,]) for test data.
- alpha (float) – Regularization strength parameter for LassoLars.

Returns
[error, prediction], or [np.nan, np.nan] if no valid data.

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*)

Generates a forecast for a single time step using LassoLars with alpha, then computes tercile probabilities based on the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Observed data (T, Y, X).
- clim_year_start (int) – Start year for the climatological reference period.
- clim_year_end (int) – End year for the climatological reference period.
- Predictor (xarray.DataArray) – Historical predictor data (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=2, T, Y, X).
- Predictor_for_year (xarray.DataArray) – Single-year predictor data (features,) or shape (1, features).
- alpha (xarray.DataArray) – Spatial map of alpha values (Y, X) for LassoLars.

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, prediction]. For a true forecast scenario, the error is typically NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA] for tercile categories.

class wass2s.was_linear_models.WAS_Lasso_Model(*alpha_range=None*, *n_clusters=5*, *nb_cores=1*, *dist_method='gamma'*)

Bases: object

WAS_Lasso_Model is a class that implements Lasso regression with hyperparameter tuning, clustering-based regional optimization, and calculation of tercile probabilities for climate prediction.

Attributes:

alpha_range
[numpy.array] Range of alpha values for Lasso regularization parameter.

n_clusters
[int] Number of clusters to use for regional optimization.

nb_cores
[int] Number of cores to use for parallel computation.

dist_method
[str] Distribution method for tercile probability calculations: one of {'gamma','t','normal','lognormal','nonparam'}.

Methods:

fit_predict(x, y, x_test, y_test, alpha):
Fits the Lasso model to the provided training data and predicts values for the test set. Returns [error, prediction].

compute_hyperparameters(predictand, predictor):
Performs clustering of the spatial grid and computes optimal alpha values for each cluster. Returns the alpha values as an xarray and the cluster assignments.

compute_model(X_train, y_train, X_test, y_test, alpha):
Computes the Lasso model prediction for the training/test datasets using the given alpha values. Utilizes Dask for parallelized processing.

calculate_tercile_probabilities(…):
Calculates tercile probabilities for a given forecast using the chosen distribution method.

compute_prob(Predictant, clim_year_start, clim_year_end, Predictor, hindcast_det):
Computes probabilistic forecasts for rainfall terciles based on climatological terciles, utilizing a hindcast and Lasso regression output.

forecast(Predictant, clim_year_start, clim_year_end, Predictor, hindcast_det,

Predictor_for_year, alpha):

Generates a forecast for a single time step using Lasso with alpha, then computes tercile probabilities based on the chosen distribution method.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)
Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)
Gamma-based method

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)
Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Clusters the spatial domain and finds the best alpha for each cluster by mean time series.

Returns

- **alpha_array** (*xarray.DataArray*) – Spatial map of best alpha values.
- **Cluster** (*xarray.DataArray*) – Cluster assignments for each grid cell.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*)

Computes Lasso predictions for spatiotemporal data using provided alpha values.

Returns
dims (output=2, Y, X) => [error, prediction].

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*, *alpha*)

Fit a Lasso model and make predictions.

Parameters

- x (ndarray) – Training data (shape: [n_samples, n_features]).
- y (ndarray) – Training targets (shape: [n_samples,]).
- x_test (ndarray) – Test data (shape: [n_features,] or [1, n_features]).
- y_test (float) – Target value(s) for test data.
- alpha (float) – Regularization parameter for Lasso.

Returns
[error, prediction], each of shape (1,) or scalar, or [np.nan, np.nan] if no valid data.

Return type
ndarray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*)

Generate a forecast for a single time (year) using Lasso with alpha, then compute tercile probabilities from the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Observed data (T, Y, X).

- clim_year_start (int) – Start year of the climatology period.
- clim_year_end (int) – End year of the climatology period.
- Predictor (xarray.DataArray) – Historical predictor data (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=2, T, Y, X).
- Predictor_for_year (xarray.DataArray) – Single-year predictor data (features,).
- alpha (xarray.DataArray) – Spatial map of alpha values (Y, X).

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, prediction]. For a real forecast scenario, the error is typically NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA].

class wass2s.was_linear_models.WAS_LinearRegression_Model(*nb_cores=1*, *dist_method='gamma'*)

Bases: object

A class to perform linear regression modeling on spatiotemporal datasets for climate prediction.

This class is designed to work with Dask and Xarray for parallelized, high-performance regression computations across large datasets with spatial and temporal dimensions. The primary methods are for fitting the model, making predictions, and calculating probabilistic predictions for climate terciles.

nb_cores

The number of CPU cores to use for parallel computation (default is 1).

Type
int, optional

dist_method

Distribution method for tercile probability calculations. One of {"t","gamma","normal","lognormal","nonparam"}. Default = "gamma".

Type
str, optional

fit_predict(*x*, *y*, *x_test*, *y_test=None*)

Fits a linear regression model, makes predictions, and calculates error if y_test is provided.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Applies the linear regression model across a dataset using parallel computation with Dask.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*)

Computes tercile probabilities for hindcast predictions over specified years.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

Generates a single-year forecast and computes tercile probabilities.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Applies linear regression across a spatiotemporal dataset in parallel.

Parameters

- X_train (xarray.DataArray) – Training predictors with dims ('T','features').
- y_train (xarray.DataArray) – Training targets with dims ('T','Y','X').
- X_test (xarray.DataArray) – Test predictors, shape ('features',) or (T, features).
- y_test (xarray.DataArray) – Test targets with dims ('Y','X'), or broadcastable.

Returns

dims ('output','Y','X'), where 'output'=[error, prediction].

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns

hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type

xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test=None*)

Fits a linear regression model to the provided training data, makes predictions on the test data, and calculates the prediction error (if y_test is provided).

Parameters

- x (array-like, shape (n_samples, n_features)) – Training data (predictors).
- y (array-like, shape (n_samples,)) – Training targets.
- x_test (array-like, shape (n_features,) or (1, n_features)) – Test data (predictors).
- y_test (float or None) – Test target value. If None, no error is computed.

Returns

If y_test is not None, returns [error, prediction]. If y_test is None, returns [prediction].

Return type

np.ndarray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

Generates a single-year forecast using linear regression, then computes tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray) – Observed data with dims (T, Y, X).
- clim_year_start (int) – Start year for climatology
- clim_year_end (int) – End year for climatology
- Predictor (xarray.DataArray) – Historical predictor data with dims (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=[error,prediction], T, Y, X).
- Predictor_for_year (xarray.DataArray) – Single-year predictor with shape (features,) or (1, features).

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, prediction]. For a true forecast, error is typically NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA].

class wass2s.was_linear_models.WAS_Ridge_Model(*alpha_range=None*, *n_clusters=5*, *nb_cores=1*, *dist_method='gamma'*)

Bases: object

A class to perform ridge regression modeling for rainfall prediction with spatial clustering and hyperparameter optimization. By Mandela HOUNGNIBO.

alpha_range

Range of alpha values to explore for ridge regression.

Type
array-like

n_clusters

Number of clusters for KMeans clustering.

Type
int

nb_cores

Number of cores to use for parallel computation.

Type
int

dist_method

Distribution method for tercile probability calculations: One of {'gamma', 't', 'normal', 'lognormal', 'nonparam'} (default = 'gamma').

Type
str

fit_predict(*x*, *y*, *x_test*, *y_test*, *alpha*)

Fits a Ridge regression model using the provided data and makes predictions.

compute_hyperparameters(*predictand*, *predictor*)

Computes optimal ridge hyperparameters (alpha values) for different clusters.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*)

Fits and predicts a Ridge model for spatiotemporal data using Dask for parallel computation.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*)

Computes the probabilities of tercile categories for rainfall prediction using the chosen distribution method.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*)

Generates a forecast for a single future time (e.g., year) and computes tercile probabilities.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Compute optimal hyperparameters (alpha) for ridge regression for different clusters.

Parameters

- predictand (xarray.DataArray) – Predictand data for clustering (dims: T, Y, X).
- predictor (xarray.DataArray or ndarray) – Predictor data for model fitting (dims: T, features).

Returns

- **alpha_array** (*xarray.DataArray*) – Spatial map of alpha values for each grid cell.
- **cluster_da** (*xarray.DataArray*) – Cluster assignment for each (Y,X).

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *alpha*)

Fit and predict ridge regression model for spatiotemporal data using Dask for parallel computation.

Parameters

- X_train (xarray.DataArray) – Training predictor data (dims: T, features).
- y_train (xarray.DataArray) – Training predictand data (dims: T, Y, X).
- X_test (xarray.DataArray) – Test predictor data (dims: features) or broadcastable.
- y_test (xarray.DataArray) – Test predictand data (dims: Y, X).
- alpha (xarray.DataArray) – Spatial map of alpha values for each grid cell.

Returns

dims (output=2, Y, X) => [error, prediction].

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using either 't', 'gamma', 'normal', 'lognormal', or 'nonparam'.

Parameters

- Predictant (xarray.DataArray) – Observed data array (T, Y, X).
- clim_year_start (int) – Start year for climatology.
- clim_year_end (int) – End year for climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with shape (output=2, T, Y, X), where output=[error, prediction].

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*, *alpha*)

Fit a ridge regression model and make predictions.

Parameters

- x (ndarray) – Training data (shape = [n_samples, n_features]).
- y (ndarray) – Target values for training data (shape = [n_samples,]).
- x_test (ndarray) – Test data (shape = [n_features,] or [1, n_features]).
- y_test (float) – Target value for test data.
- alpha (float) – Regularization strength for Ridge regression.

Returns
[error, prediction].

Return type
ndarray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha*)

Generates a ridge-based forecast for a single future time (year) using alpha values and returns both forecast + tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed data (T, Y, X).
- clim_year_start (int) – Start year of climatology.
- clim_year_end (int) – End year of climatology.
- Predictor (xarray.DataArray) – Historical predictor data (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=2, T, Y, X).
- Predictor_for_year (xarray.DataArray) – Single-year predictor data (features,).
- alpha (xarray.DataArray) – Spatial map of alpha values (Y, X).

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, prediction]. The "error" is NaN in a real forecast scenario.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA].

1.3.6 wass2s.was_eof module

class wass2s.was_eof.WAS_EOF(*n_modes=None*, *use_coslat=True*, *standardize=False*, *opti_explained_variance=None*, *L2norm=True*)

Bases: object

A class for performing Empirical Orthogonal Function (EOF) analysis using the xeofs package, with additional options for detrending and cosine latitude weighting.

Parameters

- n_modes (int, optional) – The number of EOF modes to retain. If None, the number of modes is determined by explained variance.
- use_coslat (bool, optional) – If True, applies cosine latitude weighting to account for the Earth's spherical geometry.
- standardize (bool, optional) – If True, standardizes the input data by removing the mean and dividing by the standard deviation.
- detrend (bool, optional) – If True, detrends the input data along the time dimension before performing EOF analysis.
- opti_explained_variance (float, optional) – The target cumulative explained variance (in percent) to determine the optimal number of EOF modes.
- L2norm (bool, optional) – If True, normalizes the components and scores to have L2 norm.

model

The EOF model fitted to the predictor data.

Type
xeofs.models.EOF

fit(*predictor*, *dim='T'*, *clim_year_start=None*, *clim_year_end=None*)

inverse_transform(*pcs*)

plot_EOF(*s_eofs*, *s_expvar*)

Plot the EOF spatial patterns and their explained variance.

Parameters

- s_eofs (xarray.DataArray) – The EOF spatial patterns to plot.
- s_expvar (numpy.ndarray) – The explained variance for each EOF mode.

transform(*predictor*)

1.3.7 wass2s.was_pcr module

class wass2s.was_pcr.WAS_PCR(*regression_model*, *n_modes=None*, *use_coslat=True*, *standardize=False*, *opti_explained_variance=None*, *L2norm=True*)

Bases: object

A class for performing Principal Component Regression (PCR) using EOF analysis and variable regression models.

This class integrates the WAS_EOF for dimensionality reduction through Empirical Orthogonal Function (EOF) analysis and allows the use of different regression models for predicting a target variable based on the principal components.

eof_model

The EOF analysis model used for dimensionality reduction.

Type
WAS_EOF

reg_model

A regression model (e.g., WAS_LinearRegression_Model, WAS_Ridge_Model, etc.) used for regression on the PCs.

Type
object

compute_model(*X_train*, *y_train*, *X_test*, *y_test=None*, *alpha=None*, *l1_ratio=None*, ***kwargs*)

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *alpha=None*, *l1_ratio=None*)

1.3.8 wass2s.was_cca module

class wass2s.was_cca.WAS_CCA(*n_modes=5*, *n_pca_modes=10*, *standardize=False*, *use_coslat=True*, *use_pca=True*, *dist_method='gamma'*)

Bases: object

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute the CCA model and generate hindcasts.

Parameters: - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data. - X_test: xarray DataArray for predictor testing data. - y_test: xarray DataArray for predictand testing data.

Returns: - hindcast: xarray DataArray containing predictions and errors.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_cca(*X_train*, *y_train*)

Fit the CCA model using the training data.

Parameters: - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

plot_cca_results(*X=None*, *Y=None*, *n_modes=None*, *clim_year_start=None*, *clim_year_end=None*)

Plots the CCA modes and scores.

Parameters: - X: Optional xarray DataArray for predictors. If provided, the model will be fitted using X and Y. - Y: Optional xarray DataArray for predictands. - n_modes: Number of modes to plot. If None, plots all modes.

preprocess_data(*X*, *Y*)

Preprocess the data by detrending, masking, and filling missing values.

Parameters: - X: xarray DataArray for predictors. - Y: xarray DataArray for predictands.

Returns: - X_final: Preprocessed X data. - Y_final: Preprocessed Y data.

preprocess_test_data(*X_test*, *y_test*, *X_train*, *y_train*)

Preprocess the test data.

Parameters: - X_test: xarray DataArray for predictor testing data. - y_test: xarray DataArray for predictand testing data. - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data.

Returns: - X_test_prepared: Preprocessed X test data. - y_test_prepared: Preprocessed Y test data.

class wass2s.was_cca.WAS_CCA(*n_modes=5*, *n_pca_modes=10*, *standardize=False*, *use_coslat=True*, *use_pca=True*)

Bases: object

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Calculates the probability of each tercile category.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof*)

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute the CCA model and generate hindcasts.

Parameters: - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data. - X_test: xarray DataArray for predictor testing data. - y_test: xarray DataArray for predictand testing data.

Returns: - hindcast: xarray DataArray containing predictions and errors.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Computes tercile category probabilities for hindcasts over a climatological period.

fit_cca(*X_train*, *y_train*)

Fit the CCA model using the training data.

Parameters: - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

plot_cca_results(*X=None*, *Y=None*, *n_modes=None*, *clim_year_start=None*, *clim_year_end=None*)

Plots the CCA modes and scores.

Parameters: - X: Optional xarray DataArray for predictors. If provided, the model will be fitted using X and Y. - Y: Optional xarray DataArray for predictands. - n_modes: Number of modes to plot. If None, plots all modes.

preprocess_data(*X*, *Y*)

Preprocess the data by detrending, masking, and filling missing values.

Parameters: - X: xarray DataArray for predictors. - Y: xarray DataArray for predictands.

Returns: - X_final: Preprocessed X data. - Y_final: Preprocessed Y data.

preprocess_test_data(*X_test*, *y_test*, *X_train*, *y_train*)

Preprocess the test data.

Parameters: - X_test: xarray DataArray for predictor testing data. - y_test: xarray DataArray for predictand testing data. - X_train: xarray DataArray for predictor training data. - y_train: xarray DataArray for predictand training data.

Returns: - X_test_prepared: Preprocessed X test data. - y_test_prepared: Preprocessed Y test data.

1.3.9 wass2s.was_machine_learning module

class wass2s.was_machine_learning.WAS_LogisticRegression_Model(*nb_cores=1*)

Bases: object

A logistic regression-based approach to classifying climate data into terciles and then predicting the class probabilities for new data.

static classify(*y*, *index_start*, *index_end*)

Classifies the values of a 1D array *y* into terciles. We only use a slice of y for the training/climatology period to define the 33rd and 67th percentiles.

Parameters

- y (array-like, shape (n_samples,)) – The time series of values we want to classify (e.g., rainfall).
- index_start (int) – The start and end indices defining the climatology/training window.
- index_end (int) – The start and end indices defining the climatology/training window.

Returns

- **y_class** (*array, shape (n_samples,)*) – The tercile class of each value in *y*, coded as 0 (below), 1 (middle), or 2 (above).
- **tercile_33** (*float*) – The 33rd percentile threshold used to split the data.
- **tercile_67** (*float*) – The 67th percentile threshold used to split the data.

compute_class(*Predictant*, *clim_year_start*, *clim_year_end*)

Assigns tercile classes for each point in the *Predictant* array.

Parameters

- Predictant (xarray.DataArray) – The observed variable (e.g., rainfall) with dimensions (T, Y, X).
- clim_year_start (int) – First year of the climatology period.
- clim_year_end (int) – Last year of the climatology period.

Returns

Predictant_class – The tercile class for each grid cell and time, labeled 0, 1, or 2.

Return type

xarray.DataArray

compute_model(*X_train*, *y_train*, *X_test*)

Computes logistic-regression-based class probabilities for each grid cell in *y_train*.

Parameters

- X_train (xarray.DataArray) – Predictors with dimensions (T, features).
- y_train (xarray.DataArray) – Tercile class labels with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Test predictors with dimensions (T, features).

Returns

Class probabilities (3) for each grid cell.

Return type

xarray.DataArray

fit_predict(*x*, *y*, *x_test*)

Trains a logistic regression model on (x, y) and predicts class probabilities for x_test.

Parameters

- x (array-like, shape (n_samples, n_features)) – Predictor data for training.
- y (array-like, shape (n_samples,)) – Class labels (0, 1, 2) for training.
- x_test (array-like, shape (n_features,)) – Predictor data for the forecast/unknown scenario.

Returns

preds_proba – Probability of each of the 3 tercile classes. If fewer than 3 classes were present in training, the array is padded with NaNs.

Return type

np.ndarray, shape (3,)

forecast(*Predictant*, *Predictor*, *Predictor_for_year*)

Runs the trained logistic model on a single forecast year.

Parameters

- Predictant (xarray.DataArray) – The observed variable (T, Y, X), used for classification (training).
- Predictor (xarray.DataArray) – The training predictors (T, features).
- Predictor_for_year (xarray.DataArray) – Predictors for the forecast period or year, shape (features,).

Returns
Probability of each tercile class (PB, PN, PA) for every grid cell, after removing the time dimension (because it's just one forecast).

Return type
xarray.DataArray

class wass2s.was_machine_learning.WAS_MLP(*nb_cores=1*, *dist_method='gamma'*, *n_clusters=5*, *param_grid=None*)

Bases: object

A class to perform MLP (Multi-Layer Perceptron) regression on spatiotemporal datasets for climate prediction, with hyperparameter tuning via clustering + grid search.

Parameters

- nb_cores (int) – Number of CPU cores to use for parallel computation.
- dist_method (str) – Distribution method for tercile probability calculations. One of {'gamma', 't', 'normal', 'lognormal', 'nonparam'}.
- n_clusters (int) – Number of clusters to use for KMeans.
- param_grid (dict or None) – The hyperparameter search grid for MLPRegressor. If None, a default grid is used.

nb_cores, dist_method, n_clusters, param_grid

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)
Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)
Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)
Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)
Performs KMeans clustering on the spatial mean of *predictand*, then for each cluster runs a cross-validation grid search on MLP hyperparameters using the cluster-mean time series.

Parameters

- predictand (xarray.DataArray) – Target variable with dimensions ('T', 'Y', 'X').
- predictor (xarray.DataArray) – Predictor variables with dimensions ('T', 'features').

Returns

hl_array, act_array, lr_array, cluster_da – DataArrays storing the best local hyperparameters for each grid cell (derived from cluster membership) and the cluster assignments. Note: We show example outputs for hidden_layer_sizes, activation, learning_rate_init.

You can extend this to all parameters in your *param_grid*.

Return type
xarray.DataArray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *hl_array*, *act_array*, *lr_array*, *maxiter_array*)

Runs MLP fit/predict for each (Y,X) cell in parallel, using cluster-based hyperparams.

Parameters

- X_train (xarray.DataArray) – Training predictors with dims ('T','features').
- y_train (xarray.DataArray) – Training target with dims ('T','Y','X').
- X_test (xarray.DataArray) – Test predictors, shape ('features',) or broadcastable.
- y_test (xarray.DataArray) – Test target with dims ('Y','X').
- hl_array (xarray.DataArray) – Local best hyperparameters from compute_hyperparameters.
- act_array (xarray.DataArray) – Local best hyperparameters from compute_hyperparameters.
- lr_array (xarray.DataArray) – Local best hyperparameters from compute_hyperparameters.

Returns
dims ('output', 'Y', 'X'), where 'output' = [error, prediction].

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*X_train*, *y_train*, *X_test*, *y_test*, *hl_sizes*, *activation*, *lr_init*, *maxiter*)

Trains an MLP (with local hyperparams) on the provided training data, then predicts on X_test. Returns [error, prediction].

Parameters

- X_train (np.ndarray, shape (n_samples, n_features))
- y_train (np.ndarray, shape (n_samples,))
- X_test (np.ndarray, shape (n_features,) or (1, n_features))
- y_test (float or np.nan)
- hl_sizes (str (stored as string in xarray) or None)

- activation (str)
- lr_init (float)

Returns
[error, predicted_value]

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *hl_array*, *act_array*, *lr_array*)

Generate a forecast for a single future time (e.g., future year), then compute tercile probabilities using the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Observed data with dims (T, Y, X), used for computing climatological terciles.
- clim_year_start (int) – Start year of the climatology period.
- clim_year_end (int) – End year of the climatology period.
- Predictor (xarray.DataArray) – Historical predictor data with dims (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=[error,prediction], T, Y, X). Used to compute error variance or error samples.
- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).
- hl_array (xarray.DataArray) – Hyperparameters from *compute_hyperparameters*, each with dims (Y, X) specifying local MLP settings.
- act_array (xarray.DataArray) – Hyperparameters from *compute_hyperparameters*, each with dims (Y, X) specifying local MLP settings.
- lr_array (xarray.DataArray) – Hyperparameters from *compute_hyperparameters*, each with dims (Y, X) specifying local MLP settings.

Returns

- **result_** (*xarray.DataArray*) – dims ('output','Y','X'), containing [error, prediction]. For a forecast, the "error" is generally NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => PB, PN, PA tercile probabilities.

class wass2s.was_machine_learning.WAS_PoissonRegression(*nb_cores=1*, *dist_method='gamma'*)

Bases: object

A class to perform Poisson Regression on spatiotemporal datasets for count data prediction.

This class is designed to work with Dask and Xarray for parallelized, high-performance regression computations across large datasets with spatial and temporal dimensions. The primary methods are for fitting the Poisson regression model, making predictions, and calculating probabilistic predictions for climate terciles.

nb_cores

The number of CPU cores to use for parallel computation (default is 1).

Type
int

dist_method

The method to use for tercile probability calculations, e.g. {"t", "gamma", "normal", "lognormal", "nonparam"} (default is "gamma").

Type
str

fit_predict(*x*, *y*, *x_test*, *y_test*)

Fits a Poisson regression model to the training data, predicts on test data, and computes error.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Applies the Poisson regression model across a dataset using parallel computation with Dask, returning predictions and error metrics.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*)

Computes tercile probabilities for hindcast rainfall (or count data) predictions over specified climatological years, using the chosen *dist_method*.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method for calculating tercile probabilities.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method for calculating tercile probabilities.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method for tercile probabilities.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Computes predictions for spatiotemporal data using Poisson Regression with parallel processing.

Parameters

- X_train (xarray.DataArray) – Predictor data with dimensions (T, features).
- y_train (xarray.DataArray) – Training target values (count data) with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Test data (predictors) with shape (features,) or (T, features), typically squeezed.
- y_test (xarray.DataArray) – Test target values (count data) with dimensions (Y, X) or broadcastable to (T, Y, X).

Returns
An array with a new dimension ('output', size=2) capturing [error, prediction].

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*)

Fits a Poisson regression model to the provided training data, makes predictions on the test data, and calculates the prediction error.

Parameters

- x (array-like, shape (n_samples, n_features)) – Training data (predictors).
- y (array-like, shape (n_samples,)) – Training targets (non-negative count data).
- x_test (array-like, shape (n_features,) or (1, n_features)) – Test data (predictors).
- y_test (float) – Test target value (actual counts).

Returns
[prediction_error, predicted_value]

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

Generate forecasts for a single time (e.g., future year) and compute tercile probabilities based on the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Target variable with dimensions (T, Y, X).
- clim_year_start (int) – Start year of climatology period.
- clim_year_end (int) – End year of climatology period.
- Predictor (xarray.DataArray) – Historical predictor data with dimensions (T, features).
- hindcast_det (xarray.DataArray) – Deterministic hindcast array that includes 'error' and 'prediction' over the historical period.
- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).

Returns
result_ : xarray.DataArray or numpy array with the forecast's [error, prediction]. hindcast_prob : xarray.DataArray of shape (probability=3, Y, X) with PB, PN, and PA.

Return type
tuple (**result_**, hindcast_prob)

class wass2s.was_machine_learning.WAS_PolynomialRegression(*nb_cores=1*, *degree=2*, *dist_method='gamma'*)

Bases: object

A class to perform Polynomial Regression on spatiotemporal datasets for climate prediction.

This class is designed to work with Dask and Xarray for parallelized, high-performance regression computations across large datasets with spatial and temporal dimensions. The primary methods are for fitting the polynomial regression model, making predictions, and calculating probabilistic predictions for climate terciles.

nb_cores

The number of CPU cores to use for parallel computation (default is 1).

Type
int, optional

degree

The degree of the polynomial (default is 2).

Type
int, optional

dist_method

The distribution method to compute tercile probabilities. One of {"t", "gamma", "normal", "lognormal", "nonparam"} (default is "gamma").

Type
str, optional

fit_predict(*x*, *y*, *x_test*, *y_test*)

Fits a Polynomial Regression model to the training data, predicts on test data, and computes error.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Applies the Polynomial Regression model across a dataset using parallel computation with Dask, returning predictions and error metrics.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*)

Computes tercile probabilities for hindcast rainfall predictions over specified climatological years.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

Generates a forecast for a single year (or time step) and calculates tercile probabilities using the chosen distribution method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

static calculate_tercile_probabilities_t(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Calculates the probability of each tercile category using a Student's t-based approach.

Parameters

- best_guess (array-like, shape (n_time,)) – Forecasted values.
- error_variance (float or array-like) – Error variance associated with the forecasted value.
- first_tercile (float or array-like) – Lower tercile threshold.
- second_tercile (float or array-like) – Upper tercile threshold.
- dof (int) – Degrees of freedom for the t-distribution.

Returns
pred_prob – Probability in each tercile category [Below, Normal, Above].

Return type
np.ndarray, shape (3, n_time)

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Computes predictions for spatiotemporal data using Polynomial Regression with parallel processing.

Parameters

- X_train (xarray.DataArray) – Training data (predictors) with dimensions (T, features). (It must be chunked properly in Dask, or at least be amenable to chunking.)
- y_train (xarray.DataArray) – Training target values with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Test data (predictors) with dimensions (features,) or (T, features). Typically, you'd match time steps or have a single test.
- y_test (xarray.DataArray) – Test target values with dimensions (Y, X) or broadcastable to (T, Y, X).

Returns
An array with shape (2, Y, X) after computing, where the first index is error and the second is the prediction.

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*)

Fits a Polynomial Regression model to the provided training data, makes predictions on the test data, and calculates the prediction error.

Parameters

- x (array-like, shape (n_samples, n_features)) – Training data (predictors).

- y (array-like, shape (n_samples,)) – Training targets.
- x_test (array-like, shape (n_features,) or (1, n_features)) – Test data (predictors) for which we want predictions.
- y_test (float) – Test target value (for computing error).

Returns
Array containing [prediction_error, predicted_value].

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*)

Generate forecasts for a single time (e.g., future year) and compute tercile probabilities based on the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Target variable with dimensions (T, Y, X).
- clim_year_start (int) – Start year of climatology period.
- clim_year_end (int) – End year of climatology period.
- Predictor (xarray.DataArray) – Historical predictor data with dimensions (T, features).
- hindcast_det (xarray.DataArray) – Deterministic hindcast array that includes 'error' and 'prediction' over the historical period.
- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).

Returns
result_ : xarray.DataArray or numpy array with the forecast's [error, prediction]. hindcast_prob : xarray.DataArray of shape (probability=3, Y, X) with PB, PN, and PA.

Return type
tuple (**result_**, hindcast_prob)

class wass2s.was_machine_learning.WAS_RandomForest_XGBoost_ML_Stacking(*nb_cores=1*, *dist_method='gamma'*, *n_clusters=5*, *param_grid=None*)

Bases: object

A class to perform Stacking Ensemble with RandomForest + XGBoost as base learners and a LinearRegression as the meta-model. Also supports:

- Hyperparameter tuning via KMeans + GridSearchCV
- Parallel spatiotemporal training/prediction using xarray + Dask
- Probability computation (terciles) under different distributions.

Parameters

- nb_cores (int, optional) – Number of CPU cores to use for parallel computation (default=1).
- dist_method (str, optional) – Distribution method for tercile probability calculations. One of {'gamma', 't', 'normal', 'lognormal', 'nonparam'} (default='gamma').
- n_clusters (int, optional) – Number of clusters for KMeans (default=5).

- param_grid (dict or None, optional) – The hyperparameter grid for GridSearchCV over the StackingRegressor. If None, uses a default small example grid.

Notes

In scikit-learn, you can reference parameters inside stacking base estimators with naming like "estimators__rf__n_estimators", "estimators__xgb__learning_rate", etc. The exact syntax can vary by sklearn version.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Cluster grid cells (Y,X) via KMeans on the mean of *predictand* (over T). Then for each cluster, run a cross-validation GridSearch over a StackingRegressor to find best hyperparameters. Store results in DataArrays.

Parameters

- predictand (xarray.DataArray) – Target variable with dims ('T','Y','X').
- predictor (xarray.DataArray) – Predictor variables with dims ('T','features').

Returns

- **best_param_da** (*xarray.DataArray (dtype=object or str)*) – A DataArray holding best hyperparameter sets (as strings) for each grid cell.
- **cluster_da** (*xarray.DataArray*) – The integer cluster assignment for each (Y, X).

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *best_param_da*)

Parallel fit/predict across the entire spatial domain, using cluster-based hyperparams.

Parameters

- X_train (xarray.DataArray) – Training data (predictors) with dims ('T','features').
- y_train (xarray.DataArray) – Training target with dims ('T','Y','X').
- X_test (xarray.DataArray) – Test data (predictors), shape (features,) or broadcastable across (Y, X).
- y_test (xarray.DataArray) – Test target with dims ('Y','X').
- best_param_da (xarray.DataArray) – The per-grid best_params from compute_hyperparameters (as strings).

Returns

dims ('output','Y','X'), where 'output' = [error, prediction].

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*X_train*, *y_train*, *X_test*, *y_test*, *best_params_str*)

Fit a local StackingRegressor with the best hyperparams (parsed from best_params_str), then predict on X_test, returning [error, prediction].

Parameters

- X_train (np.ndarray, shape (n_samples, n_features))
- y_train (np.ndarray, shape (n_samples,))
- X_test (np.ndarray, shape (n_features,) or (1, n_features))
- y_test (float or np.nan)
- best_params_str (str) – String of best_params (e.g. "{'estimators__rf__n_estimators':100, ...}")

Returns
[error, predicted_value]

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *best_param_da*)

Generate a forecast for a single time (e.g., future year), then compute tercile probabilities from the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Observed data with dims (T, Y, X), used for climatological terciles.
- clim_year_start (int) – Start year of the climatology.
- clim_year_end (int) – End year of the climatology.
- Predictor (xarray.DataArray) – Historical predictor data, dims (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast, dims (output=[error,prediction], T, Y, X). Used to compute error variance or error samples.

- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).
- best_param_da (xarray.DataArray) – Grid-based hyperparameters from compute_hyperparameters.

Returns

- **result_** (*xarray.DataArray*) – dims ('output','Y','X') => [error, prediction]. For a forecast, the 'error' will generally be NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => tercile probabilities PB, PN, PA.

class wass2s.was_machine_learning.WAS_RandomForest_XGBoost_Stacking_MLP(*nb_cores=1*, *dist_method='gamma'*, *n_clusters=5*, *param_grid=None*)

Bases: object

A class that performs stacking of RandomForest + XGBoost (base learners) and an MLPRegressor (meta-learner). Features:

- Hyperparameter tuning via cluster-based GridSearchCV
- Parallel spatiotemporal training/prediction
- Tercile probability calculations with various distributions

Parameters

- nb_cores (int) – Number of CPU cores to use for parallel computation.
- dist_method (str) – Distribution method for tercile probability calculations. One of {'gamma', 't', 'normal', 'lognormal', 'nonparam'}.
- n_clusters (int) – Number of clusters for KMeans.
- param_grid (dict or None) – Hyperparameter search grid for GridSearchCV. If None, a minimal default is used.

Notes

- When referencing parameters inside stacking estimators, scikit-learn uses "estimators__<est_name>__<param_name>" for base models, or "final_estimator__<param>" for the meta-model. For example:
  - "estimators__rf__n_estimators" => sets n_estimators for 'rf'
  - "estimators__xgb__max_depth" => sets max_depth for 'xgb'
  - "final_estimator__hidden_layer_sizes" => sets hidden_layer_sizes in MLPRegressor

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Gamma-based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Cluster grid cells (Y,X) via KMeans on the mean of *predictand*. For each cluster, run GridSearchCV on a StackingRegressor that has:

- RandomForest (rf) + XGBoost (xgb) as base learners
- MLPRegressor as meta-learner

Parameters

- predictand (xarray.DataArray) – Target variable with dims ('T', 'Y', 'X').
- predictor (xarray.DataArray) – Predictor variables with dims ('T','features').

Returns

- **best_param_da** (*xarray.DataArray*) – DataArray storing the best hyperparams (as strings) for each grid cell.
- **cluster_da** (*xarray.DataArray*) – The integer cluster assignment for each (Y,X).

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *best_param_da*)

Parallel training + prediction across the entire spatial domain, referencing local best_params for each grid cell.

Returns an xarray.DataArray with dim 'output' = [error, prediction].

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*X_train*, *y_train*, *X_test*, *y_test*, *best_params_str*)

For a single grid cell, parse the local best_params dict, set them on the StackingRegressor (with RF + XGB base, MLP meta), train and predict.

Returns [error, prediction].

Parameters

- X_train (np.ndarray, shape (n_samples, n_features))

- y_train (np.ndarray, shape (n_samples,))
- X_test (np.ndarray, shape (n_features,) or (1, n_features))
- y_test (float or np.nan)
- best_params_str (str) – Local best hyperparams as a stringified dict.

Returns
[error, prediction]

Return type
np.ndarray of shape (2,)

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *best_param_da*)

Generate a forecast for a single future time (e.g., future year), then compute tercile probabilities from the chosen distribution method.

Parameters

- Predictant (xarray.DataArray) – Observed data with dims (T, Y, X) used for computing climatological terciles.
- clim_year_start (int) – Start year of the climatology period.
- clim_year_end (int) – End year of the climatology period.
- Predictor (xarray.DataArray) – Historical predictor data, shape (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=[error,prediction], T, Y, X). Used to estimate error variance or error samples.
- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).
- best_param_da (xarray.DataArray) – Grid-based best hyperparams from *compute_hyperparameters*.

Returns

- **result_** (*xarray.DataArray*) – dims ('output','Y','X') => [error, prediction]. For a true forecast, the 'error' is typically NaN.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => PB, PN, PA tercile probabilities.

class wass2s.was_machine_learning.WAS_SVR(*nb_cores=1*, *n_clusters=5*, *kernel='linear'*, *gamma=None*, *C_range=[0.1, 1, 10, 100]*, *epsilon_range=[0.01, 0.1, 0.5, 1]*, *degree_range=[2, 3, 4]*, *dist_method='gamma'*)

Bases: object

A class to perform Support Vector Regression (SVR) on spatiotemporal datasets for climate prediction.

This class is designed to work with Dask and Xarray for parallelized, high-performance regression computations across large datasets with spatial and temporal dimensions. The primary methods are for fitting the SVR model, making predictions, and calculating probabilistic predictions for climate terciles.

nb_cores

The number of CPU cores to use for parallel computation (default is 1).

Type
int, optional

n_clusters
: The number of clusters to use in KMeans clustering (default is 5).

  Type
  : int, optional

kernel
: Kernel type to be used in SVR ('linear', 'poly', 'rbf', or 'all') (default is 'linear').

  Type
  : str, optional

gamma
: gamma of 'rbf' kernel function. Ignored by all other kernels, ["auto", "scale", None] by default None.

  Type
  : str, optional

C_range
: List of C values to consider during hyperparameter tuning.

  Type
  : list, optional

epsilon_range
: List of epsilon values to consider during hyperparameter tuning.

  Type
  : list, optional

degree_range
: List of degrees to consider for the 'poly' kernel during hyperparameter tuning.

  Type
  : list, optional

dist_method
: Distribution method ("gamma", "t", "normal", "lognormal", "nonparam") for probability calculations.

  Type
  : str, optional

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)
: Calculates tercile probabilities using a Student's t-based approach.

  best_guess
  : [array-like] Model predictions for each time.

  error_variance
  : [float or array-like] Variance of prediction errors.

  first_tercile, second_tercile
  : [float or array-like] The lower and upper tercile boundaries.

  dof
  : [int] Degrees of freedom for the t-distribution.

  Returns
  : **pred_prob** – Probability in each of the 3 categories (below, normal, above).

  Return type
  : np.ndarray

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Calculates tercile probabilities assuming Gamma-distributed errors.

best_guess
[array-like] Model predictions for each time.

error_variance
[float or array-like] Variance of prediction errors.

T1, T2
[float or array-like] The lower (T1) and upper (T2) tercile boundaries.

Returns
pred_prob – 3 rows for probabilities (PB, PN, PA) over time dimension.

Return type
np.ndarray

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-based method using the Gaussian CDF.

compute_hyperparameters(*predictand*, *predictor*)

Computes optimal SVR hyperparameters (C and epsilon) for each spatial cluster.

We cluster the spatial grid based on the mean values in *predictand*, then do a grid search for SVR hyperparameters on the average time series of each cluster.

Parameters

- predictand (xarray.DataArray) – Target variable with dimensions ('T', 'Y', 'X').
- predictor (xarray.DataArray) – Predictor variables with dimensions ('T', 'features').

Returns
DataArrays containing the best-fitting hyperparameters and cluster labels for each grid cell.

Return type
C_array, epsilon_array, degree_array, Cluster

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *epsilon*, *C*, *degree_array=None*)

Computes predictions for spatiotemporal data using SVR with parallel processing via Dask.

We break the data into chunks, apply the *fit_predict* function in parallel, and combine the results into an output DataArray.

Parameters

- X_train (xarray.DataArray) – Training predictors with dimensions ('T', 'features').
- y_train (xarray.DataArray) – Training targets with dimensions ('T', 'Y', 'X').
- X_test (xarray.DataArray) – Test predictors with dimensions ('features',).
- y_test (xarray.DataArray) – Test target values with dimensions ('Y', 'X').
- epsilon (xarray.DataArray) – Epsilon hyperparameters per grid point.

- C (xarray.DataArray) – C hyperparameters per grid point.
- degree_array (xarray.DataArray, optional) – Polynomial degrees per grid point (only used if kernel='poly').

Returns
Predictions & errors, stacked along a new 'output' dimension (size=2).

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns
hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type
xarray.DataArray

fit_predict(*x*, *y*, *x_test*, *y_test*, *epsilon*, *C*, *degree=None*)

Fits an SVR model to the provided training data, makes predictions on the test data, and calculates the prediction error.

We handle data-type issues (e.g., bytes input), set up the SVR with the requested parameters, fit it, and return both the error and the prediction.

Parameters

- x (array-like, shape (n_samples, n_features)) – Training predictors.
- y (array-like, shape (n_samples,)) – Training targets.
- x_test (array-like, shape (n_features,)) – Test predictors.
- y_test (float or None) – Test target value. Used to calculate error if available.
- epsilon (float) – Epsilon parameter for SVR (defines epsilon-tube).
- C (float) – Regularization parameter for SVR.
- degree (int, optional) – Degree for 'poly' kernel. Ignored if kernel != 'poly'.

Returns
A 2-element array containing [error, prediction].

Return type
np.ndarray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *epsilon*, *C*, *kernel_array*, *degree_array*, *gamma_array*)

Generates forecasts and computes probabilities for a specific year.

Parameters

- Predictant (xarray.DataArray) – Target variable (T, Y, X).
- clim_year_start (int) – Start year for climatology.
- clim_year_end (int) – End year for climatology.
- Predictor (xarray.DataArray) – Historical predictor data (T, features).
- hindcast_det (xarray.DataArray) – Deterministic hindcasts (includes 'prediction' and 'error' outputs).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target forecast year (features).
- epsilon (xarray.DataArray) – Hyperparameter grids for the model.
- C (xarray.DataArray) – Hyperparameter grids for the model.
- kernel_array (xarray.DataArray) – Hyperparameter grids for the model.
- degree_array (xarray.DataArray) – Hyperparameter grids for the model.
- gamma_array (xarray.DataArray) – Hyperparameter grids for the model.

Returns

1) The forecast results (error, prediction) for that year.
2) The corresponding tercile probabilities (PB, PN, PA).

Return type
tuple

class wass2s.was_machine_learning.WAS_Stacking_Ridge(*nb_cores=1*, *dist_method='gamma'*, *n_clusters=5*, *param_grid=None*)

Bases: object

A class that performs stacking of the following base learners:

- RandomForestRegressor (rf)
- XGBRegressor (xgb)
- MLPRegressor (mlp_base)

and uses Ridge as the meta-model.

Like the previous classes, this supports:

- Cluster-based hyperparameter tuning via KMeans + GridSearchCV
- Parallel spatiotemporal training/prediction with xarray + dask
- Various distribution methods for tercile probability calculations.

Parameters

- nb_cores (int) – Number of CPU cores to use for parallel computation.
- dist_method (str) – Distribution method for tercile probability calculations: One of {'gamma', 't', 'normal', 'lognormal', 'nonparam'}.
- n_clusters (int) – Number of clusters for KMeans (used in hyperparameter tuning).
- param_grid (Example for) – Hyperparameter grid for GridSearchCV. If None, a minimal default is used.
- param_grid –

{
“estimators__rf__n_estimators”: [50, 100], “estimators__xgb__max_depth”: [3, 6], “estimators__mlp_base__hidden_layer_sizes”: [(20,), (50, 10)], “final_estimator__alpha”: [0.1, 0.9, 5.0],

}

compute_hyperparameters(*predictand*, *predictor*)

Performs cluster-based hyperparam tuning, returns best-param DataArray.

fit_predict(*X_train*, *y_train*, *X_test*, *y_test*, *best_params_str*)

Trains a local stacking model with the best hyperparams for that grid cell, then predicts.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *best_param_da*)

Calls fit_predict(...) in parallel across all grid cells.

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Computes tercile probabilities using self.dist_method.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *best_param_da*)

Fits a forecast for a single future year (or time) and calculates tercile probabilities.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method (requires historical errors).

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

compute_hyperparameters(*predictand*, *predictor*)

Runs KMeans clustering on the mean of *predictand* (over time). Then, for each cluster, runs a cross-validation GridSearch over a stacking model with:

- RF, XGB, MLP (as base estimators)
- Ridge (as the meta-estimator).

Parameters

- predictand (xarray.DataArray) – Target variable with dims (‘T’,’Y’,’X’).
- predictor (xarray.DataArray) – Predictor variables with dims (‘T’,’features’).

Returns

- **best_param_da** (*xarray.DataArray*) – DataArray storing best hyperparams (as string) per grid cell.
- **cluster_da** (*xarray.DataArray*) – Cluster assignment for each (Y,X).

compute_model(*X_train*, *y_train*, *X_test*, *y_test*, *best_param_da*)

Parallel training + prediction across all spatial grid points. Uses local best hyperparams from best_param_da for each pixel.

Returns an xarray.DataArray with dim (‘output’,’Y’,’X’) => [error, prediction].

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using self.dist_method.

Parameters

- Predictant (xarray.DataArray (T, Y, X)) – Observed data.
- clim_year_start (int)
- clim_year_end (int) – The start and end years for the climatology.
- hindcast_det (xarray.DataArray) – Deterministic forecast with dims (output=2, T, Y, X).

Returns

hindcast_prob – dims (probability=3, T, Y, X) => [PB, PN, PA].

Return type

xarray.DataArray

fit_predict(*X_train*, *y_train*, *X_test*, *y_test*, *best_params_str*)

For a single grid cell, parse the best params, instantiate the stacking regressor, fit to local data, and predict.

Returns [error, prediction].

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *Predictor*, *hindcast_det*, *Predictor_for_year*, *best_param_da*)

Forecast for a single future year (or time) and compute tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed data with dims (T, Y, X), used for computing climatology thresholds.
- clim_year_start (int) – Start of climatology period.
- clim_year_end (int) – End of climatology period.
- Predictor (xarray.DataArray) – Historical predictor data, shape (T, features).
- hindcast_det (xarray.DataArray) – Historical deterministic forecast with dims (output=[error,prediction], T, Y, X) for computing error variance or samples.
- Predictor_for_year (xarray.DataArray) – Predictor data for the forecast year, shape (features,) or (1, features).
- best_param_da (xarray.DataArray) – Local best hyperparams from compute_hyperparameters, shape (Y, X).

Returns

- **result_** (*xarray.DataArray*) – dims (output=2, Y, X) => [error, prediction]. In a real forecast, "error" is typically NaN since we have no future observation.
- **hindcast_prob** (*xarray.DataArray*) – dims (probability=3, Y, X) => [PB, PN, PA].

1.3.10 wass2s.was_analog module

class wass2s.was_analog.WAS_Analog(*dir_to_save*, *year_start*, *year_forecast*, *reanalysis_name*, *model_name*, *method_analog='som'*, *best_prcp_models=None*, *month_of_initialization=None*, *lead_time=None*, *ensemble_mean='mean'*, *clim_year_start=None*, *clim_year_end=None*, *define_extent=None*, *index_compute=None*, *some_grid_size=(None, None)*, *some_learning_rate=0.5*, *some_neighborhood_function='gaussian'*, *some_sigma=1.0*, *dist_method='gamma'*)

Bases: object

Analog-based forecasting toolkit for seasonal climate applications.

This class orchestrates the end-to-end workflow required to build **analog ensembles** for Sea-Surface Temperature (SST) predictors and to translate them into deterministic and probabilistic rainfall forecasts over West Africa (or any user-defined domain). Three alternative analog-selection strategies are supported:

- **Self-Organising Maps** (*method_analog="som"*, default)
- **Correlation ranking** (*"cor_based"*)
- **Principal-Component (EOF) similarity** (*"pca_based"*)

In addition, the class encapsulates data acquisition (reanalysis + seasonal hindcasts), preprocessing, EOF/SOM training, tercile-based probability generation with multiple distributional options (Student-t, Gamma, Gaussian, Log-normal, non-parametric), and a set of convenient visualisation helpers.

Parameters:

dir_to_save
: [str] Directory path to save downloaded and processed data files.

year_start
: [int] Starting year for historical data.

year_forecast
: [int] Target forecast year.

reanalysis_name
: [str] Name of the reanalysis dataset (e.g., "ERA5.SST" or "NOAA.SST").

model_name
: [str] Name of the forecast model (e.g., "ECMWF_51.SST").

method_analog
: [str, optional] Analog method to use ("som", "cor_based", or "pca_based"). Default is "som".

best_prcp_models
: [list, optional] List of best precipitation models to consider.

month_of_initialization
: [int, optional] Month of initialization for forecasts. If None, uses current month.

lead_time
: [list, optional] List of lead times in months. If None, defaults to [1, 2, 3, 4, 5].

ensemble_mean
: [str, optional] Method for ensemble mean ("mean" or "median"). Default is "mean".

clim_year_start
: [int, optional] Start year for climatology period.

clim_year_end
[int, optional] End year for climatology period.

define_extent
[tuple, optional] Bounding box as (lon_min, lon_max, lat_min, lat_max) for regional analysis.

index_compute
[list, optional] List of climate indices to compute.

some_grid_size
[tuple, optional] Grid size for SOM (rows, columns). Default is (None, None) which uses automatic sizing.

some_learning_rate
[float, optional] Learning rate for SOM training. Default is 0.5.

some_neighborhood_function
[str, optional] Neighborhood function for SOM ("gaussian", "mexican_hat", etc.). Default is "gaussian".

some_sigma
[float, optional] Initial neighborhood radius for SOM. Default is 1.0.

dist_method
[str, optional] Method for probability calculation ("gamma", "t", "normal", "lognormal", "nonparam"). Default is "gamma".

Methods:

download_sst_reanalysis():
Downloads sea surface temperature reanalysis data.

download_models():
Downloads seasonal forecast model data.

standardize_timeseries():
Standardizes time series data.

calc_index():
Calculates climate indices from SST data.

compute_model():
Computes analog forecasts.

compute_prob():
Computes tercile probabilities from forecasts.

forecast():
Generates forecasts for a target year.

composite_plot():
Creates composite plots of forecast results.

Corr_Based(*predictant*, *ddd*, *itrain*, *ireference_year*)

Identify similar years using correlation-based analog method.

Finds years with high spatial correlation to the reference year's SST data.

Parameters

- predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- ddd (xarray.DataArray) – SST predictor data with dimensions (T, Y, X).
- itrain (list) – Indices of training years.

- ireference_year (list) – Indices of the reference year.

Returns
similar_years – Array of years similar to the reference year.

Return type
np.ndarray

Pca_Based(*predictant*, *ddd*, *itrain*, *ireference_year*)

Identify similar years using PCA-based analog method.

Uses EOF analysis to compute principal components and finds years with similar scores to the reference year.

Parameters

- predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- ddd (xarray.DataArray) – SST predictor data with dimensions (T, Y, X).
- itrain (list) – Indices of training years.
- ireference_year (list) – Indices of the reference year.

Returns
similar_years – Array of years similar to the reference year.

Return type
np.ndarray

SOM(*predictant*, *ddd*, *itrain*, *ireference_year*)

Identify similar years using Self-Organizing Maps (SOM).

Trains a SOM on SST data or climate indices and finds years mapped to the same Best Matching Unit (BMU) as the reference year.

Parameters

- predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- ddd (xarray.DataArray) – SST predictor data with dimensions (T, Y, X).
- itrain (list) – Indices of training years.
- ireference_year (list) – Indices of the reference year.

Returns
similar_years – Array of years similar to the reference year.

Return type
np.ndarray

calc_index(*indices*, *sst*)

Calculate climate indices from SST data.

Computes specified climate indices (e.g., NINO34, DMI) from SST data by averaging over predefined regions or computing differences for derived indices.

Parameters

- indices (list) – List of climate indices to compute (e.g., ['NINO34', 'DMI']).
- sst (xarray.DataArray) – SST data with dimensions (T, Y, X).

Returns
indices_dataset – Dataset containing computed climate indices as variables.

Return type
xarray.Dataset

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)
Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)
Gamma-based method

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Lognormal-based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)
Non-parametric method (require historical errors)

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)
Normal-based method using the Gaussian CDF.

composite_plot(*predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *plot_predictor=True*)
Create composite plots of predictors or predictands.

Plots SST composites for the forecast year and similar years, or precipitation ratios relative to climatology.

Parameters

- predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year for climatology period.
- clim_year_end (int or str) – End year for climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).
- plot_predictor (bool, optional) – If True, plot SST predictors; otherwise, plot precipitation ratios. Default is True.

Returns
similar_years – Array of similar years used in the composite.

Return type
np.ndarray

compute_model(*predictant*, *ddd*, *itrain*, *itest*)

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)
Compute tercile probabilities using either 't', 'gamma', 'normal', 'lognormal', or 'nonparam'.

Parameters:

Predictant
[xarray.DataArray] Observed data array with dimensions (T, Y, X)

clim_year_start
[int] Start year for climatology

clim_year_end
[int] End year for climatology

hindcast_det
[xarray.DataArray] Deterministic forecast (same shape as Predictant, minus M dimension)

method
[str, default = "gamma"] Method to use for calculating tercile probabilities: - "t" - "gamma" - "normal" - "lognormal" - "nonparam"

error_samples
[xarray.DataArray or None] Only required for non-parametric method, shape (ensemble, T, Y, X) or something similar.

Returns:

hindcast_prob
[xarray.DataArray] Probability for each tercile category (PB, PN, PA) with dimensions (probability=3, T, Y, X).

download_and_process(*center_variable=None*)

Download and process SST data for reanalysis and forecast.

Combines reanalysis and forecast SST data, standardizes, and applies a rolling mean.

Parameters
center_variable (str, optional) – Center-variable identifier (e.g., 'ECMWF_51.SST'). Default is None (uses class attributes).

Returns

- **concatenated_ds_st** (*xarray.DataArray*) – Standardized and processed SST data.
- **ds_shifted** (*xarray.DataArray*) – Time-shifted version of the processed SST data.

download_models(*area=[60, -180, -60, 180]*, *force_download=False*)

Download SST seasonal forecast data.

Downloads forecast data for specified models, initialization month, lead times, and spatial area.

Parameters

- area (list, optional) – Bounding box as [North, West, South, East]. Default is [60, -180, -60, 180].
- force_download (bool, optional) – If True, forces re-download even if file exists. Default is False.

Returns
ds_centers – Combined forecast dataset across models.

Return type
xarray.Dataset

download_sst_reanalysis(*area=[60, -180, -60, 180]*, *force_download=False*)

Download Sea Surface Temperature (SST) reanalysis data.

Downloads SST data from the specified reanalysis dataset for the given years and spatial area, handling both NOAA ERSST and ERA5 datasets.

Parameters

- area (list, optional) – Bounding box as [North, West, South, East]. Default is [60, -180, -60, 180].
- force_download (bool, optional) – If True, forces re-download even if file exists. Default is False.

Returns
combined_ds – Combined SST dataset for the specified years.

Return type
xarray.Dataset

forecast(*predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *forecast_year*)

plot_indices()

standardize_timeseries(*ds*)
Standardize the dataset over a specified climatology period.

1.3.11 wass2s.was_verification module

class wass2s.was_verification.WAS_Verification(*dist_method='gamma'*)
Bases: object

Verification class for evaluating weather and climate forecasts.

Provides methods to compute deterministic, probabilistic, and ensemble-based metrics, as well as visualization tools for model performance assessment.

Parameters
dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

calculate_groc(*y_true*, *y_probs*, *index_start*, *index_end*, *n_classes=3*)
Compute Generalized Receiver Operating Characteristic (GROC) score.

Averages the Area Under the Curve (AUC) for each tercile category.

Parameters
- y_true (array-like) – Observed values.
- y_probs (array-like) – Forecast probabilities with shape (n_classes, n_samples).
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.
- n_classes (int, optional) – Number of classes (default is 3 for terciles).

Returns
GROC score, ranging from 0 to 1. Returns np.nan if insufficient valid data.

Return type
float

calculate_rpss(*y_true*, *y_probs*, *index_start*, *index_end*)
Compute Ranked Probability Skill Score (RPSS).

Compares the Ranked Probability Score (RPS) of the forecast to a climatological reference.

Parameters
- y_true (array-like) – Observed values.
- y_probs (array-like) – Forecast probabilities with shape (n_classes, n_samples).
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.

Returns
RPSS score, ranging from -Inf to 1. Returns np.nan if insufficient valid data.

Return type
float

calculate_rpss_(*y_true*, *y_probs*)

Compute Ranked Probability Skill Score (RPSS) for a single grid point or sample.

Compares the Ranked Probability Score (RPS) of the forecast to a climatological reference with uniform probabilities (1/3 for each tercile).

Parameters

- y_true (array-like of shape (n_samples,)) – True class labels, already classified into terciles (0: below-normal, 1: near-normal, 2: above-normal).
- y_probs (array-like of shape (3, n_samples)) – Forecast probabilities for each tercile category.

Returns
RPSS score, ranging from -Inf to 1 (1 indicates perfect forecast skill). Returns np.nan if insufficient valid data.

Return type
float

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Calculate tercile probabilities using Student's t-distribution.

Computes probabilities for below-normal, normal, and above-normal categories.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_variance (array-like) – Variance of forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (int) – Degrees of freedom for the t-distribution.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*)

Calculate tercile probabilities using Gamma distribution.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_variance (array-like) – Variance of forecast errors.
- T1 (array-like) – First tercile threshold values.
- T2 (array-like) – Second tercile threshold values.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Calculate tercile probabilities using Lognormal distribution.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_variance (array-like) – Variance of forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Calculate tercile probabilities using a non-parametric method.

Uses historical error samples to estimate probabilities empirically.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_samples (array-like) – Historical error samples.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Calculate tercile probabilities using Normal distribution.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_variance (array-like) – Variance of forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Calculate tercile probabilities using Weibull minimum distribution.

Assumes *best_guess* as the location, *error_std* as the scale, and *dof* as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best-guess values.
- error_variance (array-like) – Variance of forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns
pred_prob – Array of shape (3, n_time) with probabilities for below, normal, and above categories.

Return type
np.ndarray

classify(*y*, *index_start*, *index_end*)

Classify data into terciles based on climatological thresholds.

Parameters

- y (array-like) – Input data to classify.
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.

Returns

•y_class
[array-like] Classified data (0: below-normal, 1: near-normal, 2: above-normal).

•tercile_33
[float] 33rd percentile threshold.

•tercile_67
[float] 67th percentile threshold.

Return type
tuple

classify_data_into_terciles(*y*, *T1*, *T2*)

Classify data into terciles based on given thresholds.

Parameters

- y (array-like) – Input data to classify.
- T1 (float) – First tercile threshold (33rd percentile).
- T2 (float) – Second tercile threshold (67th percentile).

Returns
Classified data (0: below-normal, 1: near-normal, 2: above-normal).

Return type
array-like

static classify_percent(*p*)

Classify a percentage value into predefined categories based on thresholds.

Maps a percentage ratio to one of five categories representing deviation from average.

Parameters
p (float or array-like) – Percentage value(s) to classify, typically representing a ratio to normal (%).

Returns
Category code: - 1: Well Above Average (>= 150%) - 2: Above Average (>= 110%) - 3: Near Average (>= 90%) - 4: Below Average (>= 50%) - 5: Well Below Average (< 50%)

Return type
int or array-like

compute_class(*Predictant*, *clim_year_start*, *clim_year_end*)

Compute tercile class labels for observed data.

Parameters

- Predictant (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.

Returns
Classified data with dimensions (T, Y, X), where values are 0 (below-normal), 1 (near-normal), or 2 (above-normal).

Return type
xarray.DataArray

compute_crps(*y_true*, *y_pred*, *member_dim='number'*, *dim='T'*)

Compute Continuous Ranked Probability Score (CRPS) for ensemble forecasts.

Measures the difference between the forecast ensemble distribution and observations.

Parameters

- y_true (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- y_pred (xarray.DataArray) – Ensemble forecast data with dimensions (T, number, Y, X).
- member_dim (str, optional) – Dimension name for ensemble members. Default is 'number'.
- dim (str, optional) – Dimension to compute CRPS over (typically time). Default is 'T'.

Returns
CRPS values with dimensions (Y, X).

Return type
xarray.DataArray

compute_deterministic_score(*score_func*, *obs*, *pred*)

Apply a deterministic scoring function over xarray DataArrays.

Computes the specified metric across the time dimension for each grid point.

Parameters

- score_func (callable) – Scoring function to apply (e.g., pearson_corr, kling_gupta_efficiency).
- obs (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- pred (xarray.DataArray) – Predicted data with dimensions (T, Y, X).

Returns
Score values with dimensions (Y, X).

Return type
xarray.DataArray

compute_one_year_rpss(*obs*, *prob_pred*, *clim_year_start*, *clim_year_end*, *year*)

Compute and visualize the Ranked Probability Skill Score (RPSS) for a specific year.

Applies the RPSS calculation across a gridded dataset for a single year, using tercile classifications based on a climatology period, and plots the results on a map.

Parameters

- obs (xarray.DataArray) – Observed data with dimensions (T, Y, X), where T is time, Y is latitude, and X is longitude.
- prob_pred (xarray.DataArray) – Forecast probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, near-normal, above-normal).
- clim_year_start (int or str) – Start year of the climatology period for tercile classification.
- clim_year_end (int or str) – End year of the climatology period for tercile classification.
- year (int or str) – Specific year to compute the RPSS for.

Returns
Displays a geographical plot of the RPSS values with a colorbar.

Return type
None

compute_probabilistic_score(*score_func*, *obs*, *prob_pred*, *clim_year_start*, *clim_year_end*)

Apply a probabilistic scoring function over xarray DataArrays.

Computes the specified probabilistic metric across the time dimension for each grid point.

Parameters

- score_func (callable) – Probabilistic scoring function to apply (e.g., calculate_rpss, calculate_groc).
- obs (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- prob_pred (xarray.DataArray) – Forecast probabilities with dimensions (probability, T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.

Returns
Score values with dimensions (Y, X).

Return type
xarray.DataArray

gcm_compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for GCM hindcasts.

Supports multiple distribution methods for calculating probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).

Returns
Tercile probabilities with dimensions (probability, T, Y, X), where 'probability' includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

Return type
xarray.DataArray

gcm_compute_prob_ensemble_method(*Obs_data*, *clim_year_start*, *clim_year_end*, *model_data*, *ensemble='mean'*)

Compute probabilistic forecasts for GCM ensemble data.

Uses the specified ensemble statistic to derive best-guess forecasts and computes tercile probabilities.

Parameters

- Obs_data (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- model_data (xarray.DataArray) – Model ensemble data with dimensions (T, number, Y, X).
- ensemble (str, optional) – Ensemble statistic to use ('mean' or other supported methods). Default is 'mean'.

Returns
Tercile probabilities with dimensions (probability, T, Y, X).

Return type
xarray.DataArray

gcm_validation_compute(*models_files_path*, *Obs*, *score*, *month_of_initialization*, *clim_year_start*, *clim_year_end*, *dir_to_save_roc_reliability*, *lead_time=None*, *ensemble_mean=None*, *gridded=True*)

Validate multiple General Circulation Model (GCM) forecasts using specified metrics.

Computes deterministic, probabilistic, or ensemble-based scores for GCM hindcasts by processing model data from provided file paths and comparing against observations.

Parameters

- models_files_path (dict) – Dictionary mapping model identifiers (e.g., model names) to file paths of hindcast NetCDF files.

- Obs (xarray.DataArray) – Observed data with dimensions (T, Y, X), where T is time, Y is latitude, and X is longitude.
- score (str) – Score to compute, must be a key in self.scores (e.g., 'Pearson', 'RPSS', 'ROC_CURVE', 'RELIABILITY_DIAGRAM').
- month_of_initialization (int) – Month of forecast initialization (1-12).
- clim_year_start (int or str) – Start year of the climatology period for computing tercile thresholds.
- clim_year_end (int or str) – End year of the climatology period for computing tercile thresholds.
- dir_to_save_roc_reliability (str or Path) – Directory to save ROC curves and reliability diagram plots.
- lead_time (list of int, optional) – Lead times in months to define the forecast season (e.g., [1, 2, 3] for a 3-month season). If None, no season is specified.
- ensemble_mean (str, optional) – Ensemble statistic to use for deterministic or probabilistic scores (e.g., 'mean'). If None, ensemble mean is computed automatically when required.
- gridded (bool, optional) – Whether to perform gridded validation (True) or non-gridded (not implemented). Default is True.

Returns

If score_type is 'det_score', 'prob_score', or 'ensemble_score':
Dictionary with model identifiers as keys and xarray.DataArray of scores (dimensions Y, X) as values.

If score_type is 'all_grid_prob_score' (e.g., ROC_CURVE, RELIABILITY_DIAGRAM):
None, as results are saved as plots.

If score_type is 'all_grid_det_score':
None (not implemented).

Return type
dict or None

Raises
ValueError – If the score is not recognized or if non-gridded validation is attempted (not implemented).

gcm_validation_compute_(*center_variable*, *month_of_initialization*, *lead_time*, *dir_model*, *Obs*, *year_start*, *year_end*, *clim_year_start*, *clim_year_end*, *area*, *score*, *dir_to_save_roc_reliability*, *ensemble_mean=None*, *gridded=True*)

Validate GCM forecasts using specified metrics.

Computes deterministic, probabilistic, or ensemble scores for GCM hindcasts.

Parameters

- center_variable (list of str) – List of model identifiers (e.g., 'center.variable').
- month_of_initialization (int) – Month of forecast initialization (1-12).
- lead_time (list of int) – Lead times in months for the forecast season.
- dir_model (str or Path) – Directory containing model hindcast files.
- Obs (xarray.DataArray) – Observed data with dimensions (T, Y, X).

- year_start (int) – Start year of the validation period.
- year_end (int) – End year of the validation period.
- clim_year_start (int) – Start year of the climatology period.
- clim_year_end (int) – End year of the climatology period.
- area (str) – Geographical area for validation (not used in current implementation).
- score (str) – Score to compute (e.g., 'Pearson', 'RPSS', 'ROC_CURVE').
- dir_to_save_roc_reliability (str or Path) – Directory to save ROC and reliability plots.
- ensemble_mean (str, optional) – Ensemble statistic to use if needed (e.g., 'mean'). Default is None.
- gridded (bool, optional) – Whether to perform gridded validation. Default is True.

Returns
Dictionary of model scores with keys as model identifiers and values as xarray.DataArray, or None for plotting scores (e.g., ROC_CURVE, RELIABILITY_DIAGRAM).

Return type
dict or None

get_scores_metadata()

Retrieve metadata for all available scoring metrics.

Returns
Dictionary containing score names as keys and tuples of metadata (description, min_value, max_value, score_type, colormap, function) as values.

Return type
dict

ignorance_score(*y_true*, *y_probs*, *index_start*, *index_end*)

Compute Ignorance Score based on Weijs (2010).

Measures the logarithmic loss of forecast probabilities for the true category.

Parameters

- y_true (array-like) – Observed values.
- y_probs (array-like) – Forecast probabilities with shape (n_classes, n_samples).
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.

Returns
Ignorance score, non-negative. Returns np.nan if insufficient valid data.

Return type
float

index_of_agreement(*y_true*, *y_pred*)

Compute Index of Agreement (IOA).

Measures the degree of model prediction accuracy relative to observed variability.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Returns
IOA score, ranging from 0 to 1 (1 is perfect). Returns np.nan if insufficient valid data.

Return type
float

kling_gupta_efficiency(*y_true*, *y_pred*)

Compute Kling-Gupta Efficiency (KGE) metric.

KGE combines correlation, bias, and variability ratios to assess model performance.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Returns
KGE score, ranging from -Inf to 1 (1 is perfect). Returns np.nan if insufficient valid data.

Return type
float

mean_absolute_error(*y_true*, *y_pred*)

Compute Mean Absolute Error (MAE).

Measures the average magnitude of errors in predictions.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Returns
MAE value, non-negative. Returns np.nan if insufficient valid data.

Return type
float

nash_sutcliffe_efficiency(*y_true*, *y_pred*)

Compute Nash-Sutcliffe Efficiency (NSE).

Measures the predictive skill of the model compared to the mean of observations.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Returns
NSE score, ranging from -Inf to 1 (1 is perfect). Returns np.nan if insufficient valid data.

Return type
float

pearson_corr(*y_true*, *y_pred*)

Compute Pearson Correlation Coefficient.

Measures the linear correlation between observed and predicted values.

Parameters

- y_true (array-like) – Observed values.

- y_pred (array-like) – Predicted values.

Returns
Pearson correlation coefficient, ranging from -1 to 1. Returns np.nan if insufficient valid data.

Return type
float

plot_model_score(*model_metric*, *score*, *dir_save_score*, *figure_name='WAS_MLR'*)

Plot a deterministic score on a map.

Creates a geographical plot of the specified metric for a single model.

Parameters

- model_metric (xarray.DataArray) – Score values with dimensions (Y, X).
- score (str) – Name of the score to plot (e.g., 'Pearson', 'MAE').
- dir_save_score (str or Path) – Directory to save the plot.
- figure_name (str, optional) – Prefix for the figure filename. Default is 'WAS_MLR'.

plot_models_score(*model_metrics*, *score*, *dir_save_score*)

Plot multiple model scores on a grid of maps.

Creates a subplot grid with geographical plots for each model's score.

Parameters

- model_metrics (dict) – Dictionary of model names and their corresponding score DataArrays (Y, X).
- score (str) – Name of the score to plot (e.g., 'Pearson', 'MAE').
- dir_save_score (str or Path) – Directory to save the plot.

plot_roc_curves(*modelname*, *dir_to_save_score*, *y_true*, *y_probs*, *clim_year_start*, *clim_year_end*, *n_bootstraps=200*, *ci=0.95*)

Plot ROC Curves with Confidence Intervals for probabilistic forecasts.

Visualizes the Receiver Operating Characteristic for each tercile category.

Parameters

- modelname (str) – Name of the model for labeling the plot.
- dir_to_save_score (str or Path) – Directory to save the plot.
- y_true (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- y_probs (xarray.DataArray) – Forecast probabilities with dimensions (probability, T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- n_bootstraps (int, optional) – Number of bootstrap samples for confidence intervals. Default is 200.
- ci (float, optional) – Confidence interval level (e.g., 0.95 for 95%). Default is 0.95.

ratio_to_average(*predictant*, *clim_year_start*, *clim_year_end*, *year*)

Compute and visualize the ratio of a specific year's data to the climatological mean.

Calculates the percentage ratio of the predictand for a given year relative to the mean over a climatology period, classifies it into categories, and plots the result on a geographical map.

Parameters

- predictant (xarray.DataArray) – Observed data with dimensions (T, Y, X), where T is time, Y is latitude, and X is longitude.
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- year (int or str) – Specific year to compute the ratio for.

Returns

Displays a geographical plot of the classified ratios with a custom colormap and legend.

Return type

None

reliability_diagram(*modelname*, *dir_to_save_score*, *y_true*, *y_probs*, *clim_year_start*, *clim_year_end*, *bins=array([0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7])*)

Plot Reliability Diagrams for probabilistic forecasts.

Visualizes the calibration of forecast probabilities against observed frequencies.

Parameters

- modelname (str) – Name of the model for labeling the plot.
- dir_to_save_score (str or Path) – Directory to save the plot.
- y_true (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- y_probs (xarray.DataArray) – Forecast probabilities with dimensions (probability, T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- bins (array-like, optional) – Probability bins for discretizing forecast probabilities.

reliability_score_grid(*y_true*, *y_probs*, *index_start*, *index_end*, *bins=array([0., 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.975, 1.])*)

Compute Reliability Score on a grid based on Weijs (2010).

Measures the calibration of forecast probabilities against observed frequencies.

Parameters

- y_true (array-like) – Observed values.
- y_probs (array-like) – Forecast probabilities with shape (n_classes, n_samples).
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.
- bins (array-like, optional) – Probability bins for discretizing forecast probabilities.

Returns

Reliability score. Returns np.nan if insufficient valid data.

Return type
float

resolution_and_reliability_over_all_grid(*dir_to_save_score*, *y_true*, *y_probs*, *clim_year_start*, *clim_year_end*, *bins=array([0., 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.975, 1.])*)

Compute Resolution and Reliability scores over all grid points.

Based on Weijs (2010), computes scores by aggregating across all grid points.

Parameters

- dir_to_save_score (str or Path) – Directory to save score outputs.
- y_true (xarray.DataArray) – Observed data with dimensions (T, Y, X).
- y_probs (xarray.DataArray) – Forecast probabilities with dimensions (probability, T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- bins (array-like, optional) – Probability bins for discretizing forecast probabilities.

Returns

•resolution_sum
[float] Aggregated resolution score.

•reliability_sum
[float] Aggregated reliability score.

Return type
tuple

resolution_score_grid(*y_true*, *y_probs*, *index_start*, *index_end*, *bins=array([0., 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.975, 1.])*)

Compute Resolution Score on a grid based on Weijs (2010).

Measures the ability of the forecast to distinguish between different outcomes.

Parameters

- y_true (array-like) – Observed values.
- y_probs (array-like) – Forecast probabilities with shape (n_classes, n_samples).
- index_start (int) – Start index of the climatology period.
- index_end (int) – End index of the climatology period.
- bins (array-like, optional) – Probability bins for discretizing forecast probabilities.

Returns
Resolution score, non-negative. Returns np.nan if insufficient valid data.

Return type
float

root_mean_square_error(*y_true*, *y_pred*)

Compute Root Mean Square Error (RMSE).

Measures the square root of the average squared errors in predictions.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Returns
RMSE value, non-negative. Returns np.nan if insufficient valid data.

Return type
float

taylor_diagram(*y_true*, *y_pred*)

Placeholder for Taylor Diagram visualization.

Visualizes model performance in terms of correlation, standard deviation, and RMSE.

Parameters

- y_true (array-like) – Observed values.
- y_pred (array-like) – Predicted values.

Notes

Implementation pending.

weighted_gcm_forecasts(*Obs*, *best_models*, *scores*, *lead_time*, *model_dir*, *clim_year_start*, *clim_year_end*, *variable='PRCP'*)

Generate weighted ensemble forecasts from selected GCMs based on their performance scores.

Combines hindcasts and forecasts from multiple models using weights derived from the GROC score, scales the results to match observed climatology, and computes tercile probabilities.

Parameters

- Obs (xarray.DataArray) – Observed data with dimensions (T, Y, X), used for scaling and probability calculations.
- best_models (dict) – Dictionary of model identifiers (keys) to be included in the ensemble.
- scores (dict) – Dictionary containing scores (e.g., GROC) for each model, with model identifiers as keys.
- lead_time (list of int) – Lead times in months for the forecast season (e.g., [1, 2, 3] for a 3-month season).
- model_dir (str or Path) – Directory containing hindcast and forecast NetCDF files.
- clim_year_start (int or str) – Start year of the climatology period for computing tercile thresholds.
- clim_year_end (int or str) – End year of the climatology period for computing tercile thresholds.
- variable (str, optional) – Variable name in the NetCDF files (e.g., 'PRCP' for precipitation). Default is 'PRCP'.

Returns

•hindcast_det
[xarray.DataArray] Weighted deterministic hindcast with dimensions (T, Y, X).

•hindcast_prob
[xarray.DataArray] Tercile probabilities for hindcasts with dimensions (probability, T, Y, X).

•forecast_prob
[xarray.DataArray] Tercile probabilities for forecasts with dimensions (probability, Y, X).

Return type
tuple

Notes

- Model files are expected to follow a naming convention like 'hindcast_{center}_{variable}_{init_month}_{season}_{lead}.nc'.
- The GROC score is used as the weighting metric.
- A scaling factor is applied based on the ratio of observed to hindcast mean.

1.3.12 wass2s.was_mme module

class wass2s.was_mme.BMA(*observations*, *model_predictions*, *model_names=None*, *alpha=1.0*, *error_metric='rmse'*)

Bases: object

compute_error_based_prior()
Compute either RMSE or MAE for each model vs. observations, then use exp(-alpha * error) for priors.

compute_model_posterior_probs()
Combine model priors and WAIC-based likelihood approximation to get posterior probabilities.

fit_models_pymc(*draws=2000*, *tune=1000*, *chains=4*, *target_accept=0.9*, *init='adapt_diag'*, *verbose=True*)
Fit a PyMC model for each set of predictions: y ~ offset + scale * preds + noise. Then compute WAIC and store offset, scale from posterior means.

Parameters

- draws (int) – The number of samples (in each chain) to draw from the posterior.
- tune (int) – The number of tuning (burn-in) steps.
- chains (int) – The number of chains to run.
- target_accept (float) – The target acceptance probability for the sampler.
- init (str) – The initialization method for PyMC's sampler. E.g., "adapt_diag", "jitter+adapt_diag", "advi+adapt_diag", "adapt_full", "jitter+adapt_full", "auto".

predict(*future_model_preds_list*)
Compute out-of-sample (future) predictions for each model, applying offset+scale, and then weighting by posterior probabilities.

predict_in_sample()
Compute in-sample predictions using offset + scale for each model, weighted by posterior_probs.

summary()
Print a summary for whichever error metric is used (RMSE or MAE).

class wass2s.was_mme.NonHomogeneousGaussianRegression

Bases: object

Placeholder for Non-Homogeneous Gaussian Regression model.

This class is not implemented in the provided code and serves as a placeholder for future development.

Parameters
None

class wass2s.was_mme.WAS_Min2009_ProbWeighted

Bases: object

Probability-Weighted Multi-Model Ensemble based on Min et al. (2009).

Implements a specific weighting scheme for combining multiple climate models, where weights are derived from model scores with a threshold-based transformation.

Parameters
None

compute(*rainfall*, *hdcst*, *fcst*, *scores*, *threshold=0.5*, *complete=False*)

Compute probability-weighted ensemble estimates for hindcast and forecast datasets.

Applies a weighting scheme where scores below the threshold are set to 0, and others to 1. Optionally fills missing values with unweighted averages.

Parameters

- rainfall (xarray.DataArray) – Observed rainfall data with dimensions (T, Y, X, M).
- hdcst (xarray.DataArray) – Hindcast data with dimensions (T, M, Y, X).
- fcst (xarray.DataArray) – Forecast data with dimensions (T, M, Y, X).
- scores (dict) – Dictionary mapping model names to score arrays.
- threshold (float, optional) – Threshold below which scores are set to 0. Default is 0.5.
- complete (bool, optional) – If True, fill missing values with unweighted averages. Default is False.

Returns

- **hindcast_weighted** (*xarray.DataArray*) – Weighted hindcast ensemble with dimensions (T, Y, X).
- **forecast_weighted** (*xarray.DataArray*) – Weighted forecast ensemble with dimensions (T, Y, X).

class wass2s.was_mme.WAS_mme_AdaBoost(*n_estimators=50*, *learning_rate=0.1*, *random_state=42*, *dist_method='gamma'*)

Bases: object

AdaBoost-based Multi-Model Ensemble (MME) forecasting.

This class implements a single-model forecasting approach using scikit-learn's AdaBoostRegressor for deterministic predictions, with optional tercile probability calculations using various statistical distributions.

Parameters

- n_estimators (int, optional) – Number of boosting iterations. Default is 50.
- learning_rate (float, optional) – Learning rate for boosting. Default is 0.1.

- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the AdaBoost model.

Fits the AdaBoostRegressor on training data and predicts deterministic values for the test data. Handles data stacking and NaN removal to ensure robust predictions.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).

- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns
predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant is expected to be an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the AdaBoost model.

Fits the AdaBoostRegressor on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_BMA(*obs*, *all_hdcst_models*, *all_fcst_models*, *dist_method='gamma'*, *alpha_=0.5*, *error_metric_='rmse'*)

Bases: object

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute(*draws*, *tune*, *chains*, *verbose=False*, *target_accept=0.9*, *init='jitter+adapt_diag'*)

Parameters

- draws (int) – The number of samples (in each chain) to draw from the posterior.
- tune (int) – The number of tuning (burn-in) steps.
- chains (int) – The number of chains to run.
- verbose (bool) – Show progress.
- target_accept (float) – The target acceptance probability for the sampler.
- init (str) – The initialization method for PyMC's sampler. E.g., "adapt_diag", "jitter+adapt_diag", "advi+adapt_diag", "adapt_full", "jitter+adapt_full", "auto".
- hindcasts (Runs the BMA workflow on) –

  1) compute_rmse_based_prior

  2) fit_models_pymc

  3) compute_model_posterior_probs

- (T (Returns in-sample predictions as an xarray.DataArray)
- Y

- X).

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for the hindcast using the chosen distribution method. Predictant is an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Apply BMA offset+scale weights to future forecasts, returning an xarray.DataArray (T, Y, X).

summary()

Print BMA summary information.

class wass2s.was_mme.WAS_mme_ELM(*elm_kwargs=None*, *dist_method='gamma'*)

Bases: object

Extreme Learning Machine (ELM) for Multi-Model Ensemble (MME) forecasting derived from xcast.

This class implements an Extreme Learning Machine model for deterministic forecasting, with optional tercile probability calculations using various statistical distributions.

Parameters

- elm_kwargs (dict, optional) – Keyword arguments to pass to the xcast ELM model. If None, default parameters are used: {'regularization': 10, 'hidden_layer_size': 5, 'activation': 'lin', 'preprocessing': 'none', 'n_estimators': 5}. Default is None.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.

- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns
pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type
np.ndarray

compute_model(*X_train*, *y_train*, *X_test*)

Compute deterministic hindcast using the ELM model.

Fits the ELM model on training data and predicts deterministic values for the test data. Applies regridding and drymasking to ensure data consistency.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).

Returns
result_ – Deterministic hindcast with dimensions (T, Y, X).

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for hindcast data.

Calculates probabilities for below-normal, normal, and above-normal categories using the specified distribution method, based on climatological terciles.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).

Returns
hindcast_prob – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

Return type
xarray.DataArray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross_val*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the ELM model.

Fits the ELM model on hindcast data, predicts deterministic values for the target year, and computes tercile probabilities. Applies regridding, drymasking, and standardization.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross_val (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_ELR(*elm_kwargs=None*)

Bases: object

Extended Logistic Regression (ELR) for Multi-Model Ensemble (MME) forecasting derived from xcast package.

This class implements an Extended Logistic Regression for probabilistic forecasting, directly computing tercile probabilities without requiring separate probability calculations.

Parameters

elm_kwargs (dict, optional) – Keyword arguments to pass to the xcast ELR model. If None, an empty dictionary is used. Default is None.

compute_model(*X_train*, *y_train*, *X_test*)

Compute probabilistic hindcast using the ELR model.

Fits the ELR model on training data and predicts tercile probabilities for the test data. Applies regridding and drymasking to ensure data consistency.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).

Returns

result_ – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

Return type

xarray.DataArray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *Predictor_for_year*)

Generate probabilistic forecast for a target year using the ELR model.

Fits the ELR model on hindcast data and predicts tercile probabilities for the target year. Applies regridding and drymasking to ensure data consistency.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period (not used in this method).
- clim_year_end (int or str) – End year of the climatology period (not used in this method).
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, M, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

hindcast_prob – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

Return type

xarray.DataArray

class wass2s.was_mme.WAS_mme_GA(*population_size=20*, *max_iter=50*, *crossover_rate=0.7*, *mutation_rate=0.01*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Genetic Algorithm-based Multi-Model Ensemble Weighting.

Uses a Genetic Algorithm to optimize weights for combining multiple climate models, minimizing the mean squared error (MSE) against observations. Supports tercile probability calculations.

Parameters

- population_size (int, optional) – Number of individuals in the GA population. Default is 20.
- max_iter (int, optional) – Maximum number of generations for the GA. Default is 50.
- crossover_rate (float, optional) – Probability of performing crossover. Default is 0.7.
- mutation_rate (float, optional) – Probability of mutating a gene. Default is 0.01.
- random_state (int, optional) – Seed for random number generation. Default is 42.
- dist_method (str, optional) – Distribution method for probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method for tercile probabilities.

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Train the GA and predict on test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, Y, X, M).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, Y, X, M).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Predictions with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for hindcast data using the GA-optimized weights.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).

Returns

hindcast_prob – Tercile probabilities with dimensions (probability, T, Y, X).

Return type

xarray.DataArray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Forecast for a target year and compute tercile probabilities.

Standardizes data, fits the GA, predicts for the target year, and computes probabilities.

Parameters

- Predictant (xarray.DataArray) – Historical predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, Y, X, M).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, Y, X, M).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probability forecast with dimensions (probability, T, Y, X).

class wass2s.was_mme.WAS_mme_GradientBoosting(*n_estimators=100*, *learning_rate=0.1*, *max_depth=3*, *random_state=42*, *dist_method='gamma'*)

Bases: object

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Fit the GradientBoostingRegressor on the training data and predict on X_test. The input data (xarray DataArrays) are assumed to have dimensions including 'T', 'Y', 'X'. The predictions are then reshaped back to (T, Y, X).

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant is expected to be an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Forecast method using a single MLP model. Steps:

- Standardize the predictor for the target year using hindcast climatology.
- Fit the MLP (if not already fitted) on standardized hindcast data.
- Predict for the target year.
- Reconstruct predictions into original (T, Y, X) shape.
- Reverse the standardization.
- Compute tercile probabilities using the chosen distribution.

class wass2s.was_mme.WAS_mme_LGBM_Boosting(*n_estimators=100*, *learning_rate=0.1*, *max_depth=-1*, *random_state=42*, *dist_method='gamma'*)

Bases: object

LightGBM-based Multi-Model Ensemble (MME) forecasting.

This class implements a single-model forecasting approach using LightGBM's LGBMRegressor for deterministic predictions, with optional tercile probability calculations using various statistical distributions.

Parameters

- n_estimators (int, optional) – Number of boosting iterations. Default is 100.
- learning_rate (float, optional) – Learning rate for boosting. Default is 0.1.
- max_depth (int, optional) – Maximum tree depth (-1 for no limit). Default is -1.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and

- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the LightGBM model.

Fits the LGBMRegressor on training data and predicts deterministic values for the test data. Handles data stacking and NaN removal to ensure robust predictions.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant is expected to be an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the LightGBM model.

Fits the LGBMRegressor on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).

- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_MLP(*hidden_layer_sizes=(10, 5)*, *activation='relu'*, *solver='adam'*, *max_iter=200*, *alpha=0.01*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Multi-Layer Perceptron (MLP) for Multi-Model Ensemble (MME) forecasting.

This class implements a Multi-Layer Perceptron model using scikit-learn's MLPRegressor for deterministic forecasting, with optional tercile probability calculations using various statistical distributions.

Parameters

- hidden_layer_sizes (tuple, optional) – Sizes of the hidden layers in the MLP (e.g., (10, 5)). Default is (10, 5).
- activation (str, optional) – Activation function for the hidden layers ('identity', 'logistic', 'tanh', 'relu'). Default is 'relu'.
- solver (str, optional) – Optimization algorithm ('lbfgs', 'sgd', 'adam'). Default is 'adam'.
- max_iter (int, optional) – Maximum number of iterations for the solver. Default is 200.
- alpha (float, optional) – L2 regularization parameter. Default is 0.01.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the MLP model.

Fits the MLPRegressor on training data and predicts deterministic values for the test data. Handles data stacking and NaN removal.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for the hindcast using the chosen distribution method. Predictant is an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the MLP model.

Fits the MLPRegressor on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.

- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_StackXGboost_Ml(*n_estimators=100*, *max_depth=3*, *learning_rate=0.1*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with XGBoost base model and Linear Regression meta-model for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using XGBRegressor as the base model, with a LinearRegression meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- n_estimators (int, optional) – Number of gradient boosted trees in XGBoost. Default is 100.
- max_depth (int, optional) – Maximum depth of each tree in XGBoost. Default is 3.
- learning_rate (float, optional) – Boosting learning rate for XGBoost. Default is 0.1.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Train a stacking regressor with XGBoost as the base model and Linear Regression as the meta-model, then generate deterministic predictions on X_test.

Parameters

- X_train (xarray.DataArray) – Predictor training data with dimensions (T, Y, X, M) or (T, Y, X).
- y_train (xarray.DataArray) – Predictand training data with dimensions (T, Y, X, M) or (T, Y, X).
- X_test (xarray.DataArray) – Predictor testing data with dimensions (T, Y, X, M) or (T, Y, X).
- y_test (xarray.DataArray) – Predictand testing data with dimensions (T, Y, X, M) or (T, Y, X).

Returns

predicted_da – The predictions over X_test in the original grid shape.

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. This method extracts the climatology terciles from Predictant over the period [clim_year_start, clim_year_end], computes an error variance (or uses error samples), and then applies the chosen probability function.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Forecast method that uses the trained stacking ensemble to predict for a new year, then computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Historical predictand data.
- clim_year_start (int or str) – Start of the climatology period (e.g., 1981).
- clim_year_end (int or str) – End of the climatology period (e.g., 2010).
- hindcast_det (xarray.DataArray) – Deterministic hindcasts used for training (predictors).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcasts for cross-validation (for error estimation).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target forecast year.

Returns

- **result_da** (*xarray.DataArray*) – The deterministic forecast for the target year.
- **hindcast_prob** (*xarray.DataArray*) – Tercile probability forecast for the target year.

class wass2s.was_mme.WAS_mme_Stack_KNN_Tree_SVR(*n_neighbors=5*, *tree_max_depth=None*, *svr_C=1.0*, *svr_kernel='rbf'*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with K-Nearest Neighbors and Decision Tree base models, and SVR meta-model for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using KNeighborsRegressor and DecisionTreeRegressor as base models, with an SVR meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- n_neighbors (int, optional) – Number of neighbors for K-Nearest Neighbors. Default is 5.
- tree_max_depth (int or None, optional) – Maximum depth for the Decision Tree (None for no limit). Default is None.
- svr_C (float, optional) – Regularization parameter for SVR. Default is 1.0.
- svr_kernel (str, optional) – Kernel type for SVR ('linear', 'poly', 'rbf', 'sigmoid'). Default is 'rbf'.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.

- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns
pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type
np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the stacking ensemble of KNN and Decision Tree with an SVR meta-model.

Fits the stacking ensemble (KNeighborsRegressor and DecisionTreeRegressor base models with an SVR meta-model) on training data and predicts deterministic values for the test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns
predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant is expected to be an xarray DataArray with dimensions (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the stacking ensemble.

Fits the stacking ensemble (KNN and Decision Tree base models with an SVR meta-model) on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_Stack_Lasso_RF_MLP(*lasso_alpha=0.01*, *n_estimators=100*, *mlp_hidden_layer_sizes=(10, 5)*, *mlp_activation='relu'*, *mlp_solver='adam'*, *mlp_max_iter=200*, *mlp_alpha=0.01*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with Lasso and Random Forest base models, and MLP meta-model for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using Lasso and RandomForestRegressor as base models, with an MLPRegressor meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- lasso_alpha (float, optional) – Regularization strength for the Lasso base model. Default is 0.01.
- n_estimators (int, optional) – Number of trees in the Random Forest base model. Default is 100.
- mlp_hidden_layer_sizes (tuple, optional) – Sizes of the hidden layers for the MLP meta-model (e.g., (10, 5)). Default is (10, 5).
- mlp_activation (str, optional) – Activation function for the MLP ('identity', 'logistic', 'tanh', 'relu'). Default is 'relu'.
- mlp_solver (str, optional) – Optimization algorithm for the MLP ('lbfgs', 'sgd', 'adam'). Default is 'adam'.
- mlp_max_iter (int, optional) – Maximum number of iterations for the MLP solver. Default is 200.
- mlp_alpha (float, optional) – L2 regularization parameter for the MLP. Default is 0.01.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the stacking ensemble of Lasso and Random Forest with an MLP meta-model.

Fits the stacking ensemble (Lasso and Random Forest base models with an MLPRegressor meta-model) on training data and predicts deterministic values for the test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant should be an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the stacking ensemble.

Fits the stacking ensemble (Lasso and Random Forest base models with an MLPRegressor meta-model) on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_Stack_MLP_Ada_Ridge(*hidden_layer_sizes=(10, 5)*, *activation='relu'*, *max_iter=200*, *solver='adam'*, *mlp_alpha=0.01*, *n_estimators_adaboost=50*, *ridge_alpha=1.0*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with MLP and AdaBoost base models, and Ridge meta-model for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using MLPRegressor and AdaBoostRegressor as base models, with a Ridge meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- hidden_layer_sizes (tuple, optional) – Sizes of the hidden layers for the MLP base model (e.g., (10, 5)). Default is (10, 5).
- activation (str, optional) – Activation function for the MLP ('identity', 'logistic', 'tanh', 'relu'). Default is 'relu'.
- max_iter (int, optional) – Maximum number of iterations for the MLP solver. Default is 200.
- solver (str, optional) – Optimization algorithm for the MLP ('lbfgs', 'sgd', 'adam'). Default is 'adam'.
- mlp_alpha (float, optional) – L2 regularization parameter for the MLP. Default is 0.01.
- n_estimators_adaboost (int, optional) – Number of boosting iterations for AdaBoost. Default is 50.

- ridge_alpha (float, optional) – Regularization strength for the Ridge meta-model. Default is 1.0.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the stacking ensemble of MLP and AdaBoost with a Ridge meta-model.

Fits the stacking ensemble (MLP and AdaBoost base models with a Ridge meta-model) on training data and predicts deterministic values for the test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. Predictant is expected to be an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the stacking ensemble.

Fits the stacking ensemble (MLP and AdaBoost base models with a Ridge meta-model) on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_Stack_MLP_RF(*hidden_layer_sizes=(10, 5)*, *activation='relu'*, *max_iter=200*, *solver='adam'*, *random_state=42*, *alpha=0.01*, *n_estimators=100*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with MLP and Random Forest for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using MLPRegressor and RandomForestRegressor as base models, with a LinearRegression meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- hidden_layer_sizes (tuple, optional) – Sizes of the hidden layers for the MLP base model (e.g., (10, 5)). Default is (10, 5).
- activation (str, optional) – Activation function for the MLP ('identity', 'logistic', 'tanh', 'relu'). Default is 'relu'.
- max_iter (int, optional) – Maximum number of iterations for the MLP solver. Default is 200.
- solver (str, optional) – Optimization algorithm for the MLP ('lbfgs', 'sgd', 'adam'). Default is 'adam'.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- alpha (float, optional) – L2 regularization parameter for the MLP. Default is 0.01.
- n_estimators (int, optional) – Number of trees in the Random Forest base model. Default is 100.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns
pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type
np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the stacking ensemble of MLP and Random Forest.

Fits the stacking ensemble (MLP and Random Forest base models with a LinearRegression meta-model) on training data and predicts deterministic values for the test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns
predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type
xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities using the chosen distribution method. This method extracts the climatology terciles from Predictant over the period [clim_year_start, clim_year_end], computes an error variance (or uses error samples), and then applies the chosen probability function.

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Forecast method that uses the trained stacking ensemble to predict for a new year, then computes tercile probabilities.

class wass2s.was_mme.WAS_mme_Stack_RF_GB_Ridge(*n_estimators_rf=100*, *max_depth_rf=None*, *n_estimators_gb=100*, *learning_rate_gb=0.1*, *ridge_alpha=1.0*, *random_state=42*, *dist_method='gamma'*)

Bases: object

Stacking ensemble with Random Forest and Gradient Boosting base models, and Ridge meta-model for Multi-Model Ensemble (MME) forecasting.

This class implements a stacking ensemble using RandomForestRegressor and GradientBoostingRegressor as base models, with a Ridge meta-model, for deterministic forecasting and optional tercile probability calculations.

Parameters

- n_estimators_rf (int, optional) – Number of trees in the Random Forest base model. Default is 100.
- max_depth_rf (int or None, optional) – Maximum depth for Random Forest trees (None for no limit). Default is None.
- n_estimators_gb (int, optional) – Number of boosting iterations for Gradient Boosting. Default is 100.
- learning_rate_gb (float, optional) – Learning rate for Gradient Boosting. Default is 0.1.

- ridge_alpha (float, optional) – Regularization strength for the Ridge meta-model. Default is 1.0.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the stacking ensemble of Random Forest and Gradient Boosting with a Ridge meta-model.

Fits the stacking ensemble (Random Forest and Gradient Boosting base models with a Ridge meta-model) on training data and predicts deterministic values for the test data.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for hindcasts based on the chosen distribution. Predictant is an xarray DataArray with dims (T, Y, X).

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the stacking ensemble.

Fits the stacking ensemble (Random Forest and Gradient Boosting base models with a Ridge meta-model) on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).
- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

class wass2s.was_mme.WAS_mme_Weighted(*equal_weighted=False*, *dist_method='gamma'*, *metric='GROC'*, *threshold=0.5*)

Bases: object

Weighted Multi-Model Ensemble (MME) for climate forecasting.

This class implements a weighted ensemble approach for combining multiple climate models, supporting both equal weighting and score-based weighting. It also provides methods for computing tercile probabilities using various statistical distributions.

Parameters

- equal_weighted (bool, optional) – If True, use equal weights for all models; otherwise, use score-based weights. Default is False.
- dist_method (str, optional) – Statistical distribution for probability calculations ('t', 'gamma', 'normal', 'lognormal', 'weibull_min', 'nonparam'). Default is 'gamma'.
- metric (str, optional) – Performance metric for weighting ('MAE', 'Pearson', 'GROC'). Default is 'GROC'.
- threshold (float, optional) – Threshold for score transformation. Default is 0.5.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute(*rainfall*, *hdcst*, *fcst*, *scores*, *complete=False*)

Compute weighted hindcast and forecast using model scores.

This method calculates weighted averages of hindcast and forecast data based on model scores. If *complete* is True, missing values are filled with unweighted averages.

Parameters

- rainfall (xarray.DataArray) – Observed rainfall data with dimensions (T, Y, X, M).
- hdcst (xarray.DataArray) – Hindcast data with dimensions (T, M, Y, X).
- fcst (xarray.DataArray) – Forecast data with dimensions (T, M, Y, X).
- scores (dict) – Dictionary mapping model names to score arrays.
- complete (bool, optional) – If True, fill missing values with unweighted averages. Default is False.

Returns

- **hindcast_det** (*xarray.DataArray*) – Weighted hindcast data with dimensions (T, Y, X).
- **forecast_det** (*xarray.DataArray*) – Weighted forecast data with dimensions (T, Y, X).

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for hindcast data.

Calculates tercile probabilities based on the specified distribution method, using climatological terciles derived from the predictand data.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).

Returns
hindcast_prob – Tercile probabilities with dimensions (probability, T, Y, X).

Return type
xarray.DataArray

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *forecast_det*)

transform_score(*score_array*)

Transform score array based on the chosen metric and threshold.

For 'MAE', scores below the threshold are set to 1, others to 0. For 'Pearson' or 'GROC', scores above the threshold are set to 1, others to 0.

Parameters
score_array (xarray.DataArray) – Score array to transform.

Returns
transformed_score – Transformed score array with binary weights.

Return type
xarray.DataArray

class wass2s.was_mme.WAS_mme_XGBoosting(*n_estimators=100*, *learning_rate=0.1*, *max_depth=3*, *random_state=42*, *dist_method='gamma'*)

Bases: object

XGBoost-based Multi-Model Ensemble (MME) forecasting.

This class implements a single-model forecasting approach using XGBoost's XGBRegressor for deterministic predictions, with optional tercile probability calculations using various statistical distributions.

Parameters

- n_estimators (int, optional) – Number of boosting rounds. Default is 100.
- learning_rate (float, optional) – Learning rate for boosting. Default is 0.1.
- max_depth (int, optional) – Maximum tree depth for base learners. Default is 3.
- random_state (int, optional) – Seed for random number generation for reproducibility. Default is 42.
- dist_method (str, optional) – Distribution method for tercile probability calculations ('t', 'gamma', 'nonparam', 'normal', 'lognormal', 'weibull_min'). Default is 'gamma'.

static calculate_tercile_probabilities(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Student's t-based method

static calculate_tercile_probabilities_gamma(*best_guess*, *error_variance*, *T1*, *T2*, *dof=None*)

Gamma-distribution based method.

static calculate_tercile_probabilities_lognormal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Lognormal-distribution based method.

static calculate_tercile_probabilities_nonparametric(*best_guess*, *error_samples*, *first_tercile*, *second_tercile*)

Non-parametric method using historical error samples.

static calculate_tercile_probabilities_normal(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*)

Normal-distribution based method.

static calculate_tercile_probabilities_weibull_min(*best_guess*, *error_variance*, *first_tercile*, *second_tercile*, *dof*)

Weibull minimum-based method.

Here, we assume:

- best_guess is used as the location,
- error_std (sqrt(error_variance)) as the scale, and
- dof (degrees of freedom) as the shape parameter.

Parameters

- best_guess (array-like) – Forecast or best guess values.
- error_variance (array-like) – Variance associated with forecast errors.
- first_tercile (array-like) – First tercile threshold values.
- second_tercile (array-like) – Second tercile threshold values.
- dof (float or array-like) – Shape parameter for the Weibull minimum distribution.

Returns

pred_prob – A 3 x n_time array with probabilities for being below the first tercile, between the first and second tercile, and above the second tercile.

Return type

np.ndarray

compute_model(*X_train*, *y_train*, *X_test*, *y_test*)

Compute deterministic hindcast using the XGBoost model.

Fits the XGBRegressor on training data and predicts deterministic values for the test data. Handles data stacking and NaN removal to ensure robust predictions.

Parameters

- X_train (xarray.DataArray) – Training predictor data with dimensions (T, M, Y, X).
- y_train (xarray.DataArray) – Training predictand data with dimensions (T, Y, X).
- X_test (xarray.DataArray) – Testing predictor data with dimensions (T, M, Y, X).
- y_test (xarray.DataArray) – Testing predictand data with dimensions (T, Y, X).

Returns

predicted_da – Deterministic hindcast with dimensions (T, Y, X).

Return type

xarray.DataArray

compute_prob(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*)

Compute tercile probabilities for hindcast data.

Calculates probabilities for below-normal, normal, and above-normal categories using the specified distribution method, based on climatological terciles.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data with dimensions (T, Y, X).

Returns

- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).
- *Compute tercile probabilities using the chosen distribution method.*
- *Predictant is expected to be an xarray DataArray with dims (T, Y, X).*

forecast(*Predictant*, *clim_year_start*, *clim_year_end*, *hindcast_det*, *hindcast_det_cross*, *Predictor_for_year*)

Generate deterministic and probabilistic forecast for a target year using the XGBoost model.

Fits the XGBRegressor on standardized hindcast data, predicts deterministic values for the target year, reverses standardization, and computes tercile probabilities.

Parameters

- Predictant (xarray.DataArray) – Observed predictand data with dimensions (T, Y, X) or (T, M, Y, X).
- clim_year_start (int or str) – Start year of the climatology period.
- clim_year_end (int or str) – End year of the climatology period.
- hindcast_det (xarray.DataArray) – Deterministic hindcast data for training with dimensions (T, M, Y, X).

- hindcast_det_cross (xarray.DataArray) – Deterministic hindcast data for error estimation with dimensions (T, Y, X).
- Predictor_for_year (xarray.DataArray) – Predictor data for the target year with dimensions (T, M, Y, X).

Returns

- **result_da** (*xarray.DataArray*) – Deterministic forecast with dimensions (T, Y, X).
- **hindcast_prob** (*xarray.DataArray*) – Tercile probabilities with dimensions (probability, T, Y, X), where probability includes ['PB', 'PN', 'PA'] (below-normal, normal, above-normal).

wass2s.was_mme.myfill(*all_model_fcst*, *obs*)

Fill missing values in forecast data using random samples from observations.

This function fills NaN values in the forecast data by randomly sampling values from the observed rainfall data along the time dimension.

Parameters

- all_model_fcst (xarray.DataArray) – Forecast data with dimensions (T, M, Y, X) containing possible NaN values.
- obs (xarray.DataArray) – Observed rainfall data with dimensions (T, Y, X) used for filling NaNs.

Returns

da_filled_random – Forecast data with NaN values filled using random samples from observations.

Return type

xarray.DataArray

wass2s.was_mme.process_datasets_for_mme(*rainfall*, *hdcsted=None*, *fcsted=None*, *gcm=True*, *agroparam=False*, *Prob=False*, *ELM_ELR=False*, *dir_to_save_model=None*, *best_models=None*, *scores=None*, *year_start=None*, *year_end=None*, *model=True*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*, *score_metric='GROC'*)

Process hindcast and forecast datasets for a multi-model ensemble.

This function loads, interpolates, and concatenates hindcast and forecast datasets from various sources (GCMs, agroparameters, or others) to prepare them for a multi-model ensemble. It supports different score metrics and configurations for probabilistic or deterministic outputs.

Parameters

- rainfall (xarray.DataArray) – Observed rainfall data used for interpolation and masking.
- hdcsted (dict, optional) – Dictionary of hindcast datasets for different models.
- fcsted (dict, optional) – Dictionary of forecast datasets for different models.
- gcm (bool, optional) – If True, process data as GCM data. Default is True.
- agroparam (bool, optional) – If True, process data as agroparameter data. Default is False.
- Prob (bool, optional) – If True, process data as probabilistic forecasts. Default is False.
- ELM_ELR (bool, optional) – If True, use ELM_ELR configuration for dimension renaming. Default is False.

- dir_to_save_model (str, optional) – Directory path to load model data.
- best_models (list, optional) – List of model names to include in the ensemble.
- scores (dict, optional) – Dictionary containing model scores, with the key specified by *score_metric*.
- year_start (int, optional) – Starting year for the data range.
- year_end (int, optional) – Ending year for the data range.
- model (bool, optional) – If True, treat data as model-based. Default is True.
- month_of_initialization (int, optional) – Month when the forecast is initialized.
- lead_time (int, optional) – Forecast lead time in months.
- year_forecast (int, optional) – Year for which the forecast is generated.
- score_metric (str, optional) – Metric used to organize scores (e.g., 'Pearson', 'MAE', 'GROC'). Default is 'GROC'.

Returns

- **all_model_hdcst** (*xarray.DataArray*) – Concatenated hindcast data across models.
- **all_model_fcst** (*xarray.DataArray*) – Concatenated forecast data across models.
- **obs** (*xarray.DataArray*) – Observed rainfall data expanded with a model dimension and masked.
- **scores_organized** (*dict*) – Dictionary of organized scores for selected models.

wass2s.was_mme.process_datasets_for_mme(*rainfall*, *hdcsted=None*, *fcsted=None*, *gcm=True*, *agroparam=False*, *ELM_ELR=False*, *dir_to_save_model=None*, *best_models=None*, *scores=None*, *year_start=None*, *year_end=None*, *model=True*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

1.3.13 wass2s.utils module

wass2s.utils.anomalize_timeseries(*ds*, *clim_year_start=None*, *clim_year_end=None*)

wass2s.utils.build_iridl_url_ersst(*year_start: int*, *year_end: int*, *bbox: list*, *run_avg: int = 3*, *month_start: str = 'Jan'*, *month_end: str = 'Dec'*)

Build a parameterized IRIDL URL for NOAA/ERSST, using a numeric bounding box of the form [North, West, South, East].

e.g. area = [10, -15, -5, 15].

IRIDL wants Y/(south)/(north)/…, X/(west)/(east)/…, so we reorder:

- south = area[2]
- north = area[0]
- west = area[1]
- east = area[3]

If run_avg is provided, we'll append T/<run_avg>/runningAverage/.

wass2s.utils.compute_other_indices(*dir_to_data*, *indices_dict*, *variables_list*, *year_start*, *year_end*, *season*, *clim_year_start=None*, *clim_year_end=None*, *model=False*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

Compute indices for other variables.

wass2s.utils.compute_sst_indices(*dir_to_data*, *indices*, *variables_list*, *year_start*, *year_end*, *season*, *clim_year_start=None*, *clim_year_end=None*, *others_zone=None*, *model=False*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

Compute SST indices for reanalysis or model data.

wass2s.utils.decode_cf(*ds*, *time_var*)

Decodes time dimension to CFTime standards.

wass2s.utils.download_file(*url*, *local_path*, *force_download=False*, *chunk_size=8192*, *timeout=120*)

wass2s.utils.find_best_distribution_grid(*rainfall*, *distribution_map=None*)

Apply a function across the rainfall DataArray (assumed to have a 'T' dimension) to determine the best-fitting distribution, returning a grid of numeric codes.

Parameters

- rainfall (xarray.DataArray) – Precipitation data with a time dimension 'T' and additional spatial dimensions.
- distribution_map (dict, optional) –

A mapping of distribution names to numeric codes. Defaults to:

{
'norm': 1, 'lognorm': 2, 'expon': 3, 'gamma': 4, 'weibull_min': 5
}

Returns

best_fit_da – An array of the same spatial dimensions as rainfall with the best-fitting distribution's numeric code at each grid cell.

Return type

xarray.DataArray

wass2s.utils.fix_time_coord(*ds*, *seas*)

wass2s.utils.get_best_models(*center_variable*, *scores*, *metric='MAE'*, *threshold=None*, *top_n=6*, *gcm=False*, *agroparam=False*)

wass2s.utils.load_gridded_predictor(*dir_to_data*, *variables_list*, *year_start*, *year_end*, *season=None*, *model=False*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

Load gridded predictor data for reanalysis or model.

wass2s.utils.parse_variable(*variables_list*)

Extract center and variable names from the variables list.

wass2s.utils.plot_date(*A*)

Plots 'A' on a map, interpreting the data values as offsets from 2024-01-01. The colorbar ticks are then converted to calendar dates.

wass2s.utils.plot_map(*extent*, *title='Map'*, *sst_indices=None*, *fig_size=(10, 8)*)

Plots a map with specified geographic extent and optionally adds SST index boxes.

Parameters: - extent: list of float, specifying [west, east, south, north] - title: str, title of the map - sst_indices: dict, optional dictionary containing SST index information

wass2s.utils.plot_prob_forecasts(*dir_to_save*, *forecast_prob*, *model_name*, *labels=['Below-Normal', 'Near-Normal', 'Above-Normal']*, *reverse_cmap=True*)

wass2s.utils.plot_prob_forecasts_(*dir_to_save*, *forecast_prob*, *model_name*)

wass2s.utils.plot_tercile(*A*)

wass2s.utils.predictant_mask(*data*)

wass2s.utils.prepare_predictand(*dir_to_save_Obs*, *variables_obs*, *year_start*, *year_end*, *season=None*, *ds=True*, *daily=False*)

Prepare the predictand dataset.

wass2s.utils.process_model_for_other_params(*agmParamModel*, *dir_to_save*, *hdcst_file_path*, *fcst_file_path*, *obs_hdcst*, *obs_fcst_year*, *month_of_initialization*, *year_start*, *year_end*, *year_forecast*, *nb_cores=2*, *agrometparam='Onset'*)

wass2s.utils.retrieve_several_zones_for_PCR(*dir_to_data*, *indices_dict*, *variables_list*, *year_start*, *year_end*, *season*, *clim_year_start=None*, *clim_year_end=None*, *model=False*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

Compute indices for other variables.

wass2s.utils.retrieve_single_zone_for_PCR(*dir_to_data*, *indices_dict*, *variables_list*, *year_start*, *year_end*, *season=None*, *clim_year_start=None*, *clim_year_end=None*, *model=False*, *month_of_initialization=None*, *lead_time=None*, *year_forecast=None*)

Compute indices for other variables.

wass2s.utils.reverse_standardize(*ds_st*, *ds*, *clim_year_start=None*, *clim_year_end=None*)

wass2s.utils.save_hindcast_and_forecasts(*dir_to_save*, *data*, *variable*, *forecast=None*, *deterministic=True*)

wass2s.utils.save_validation_score(*dir_to_save*, *data*, *metric*, *model_name*)

wass2s.utils.standardize_timeseries(*ds*, *clim_year_start=None*, *clim_year_end=None*)

Standardize the dataset over a specified climatology period.

wass2s.utils.to_iridl_lat(*lat: float*) → str

Convert numeric latitude to IRIDL lat string: +10 -> ‘10N’, -5 -> ‘5S’.

wass2s.utils.to_iridl_lon(*lon: float*) → str

Convert numeric longitude to IRIDL lon string: +15 -> ‘15E’, -15 -> ‘15W’.

wass2s.utils.trend_data(*data*)

trend the data using ExtendedEOF if detrending is enabled.

Parameters: - data: xarray DataArray we want to know trend.

Returns: - data_trended: trended xarray DataArray.

wass2s.utils.verify_station_network(*df_filtered*, *extent*, *map_name='Rain-gauge network'*)

Verify the station network by plotting the locations of the stations on a map.

PYTHON MODULE INDEX

W

INDEX

D

E

F

P

R

S

T

V

W