climpred

Jan 07, 2020

Getting Started

1 Version 1 Release **3**

2 Installation **5**

Bibliography **101**

Index **103**

CHAPTER 1

Version 1 Release

We currently only support annual forecasts, but our focus is to support sub-annual (*e.g.*, seasonal, monthly, weekly, daily) in our next major release (v2.0.0). We provide a host of deterministic metrics, as well as some probabilistic metrics, although the latter have not been tested rigorously. We support both perfect-model and hindcast prediction ensembles, and provide `PerfectModelEnsemble` and `HindcastEnsemble` classes to make analysis easier.

See quick start and our examples to get started.

CHAPTER 2

Installation

You can install the latest release of `climpred` using `pip` or `conda`:

```
pip install climpred
```

```
conda install -c conda-forge climpred
```

You can also install the bleeding edge (pre-release versions) by cloning this repository and running `pip install . --upgrade` in the main directory

Getting Started

- *Overview: Why climpred?*
- *Scope of climpred*
- *Quick Start*
- *Examples*

2.1 Overview: Why climpred?

There are many packages out there related to computing metrics on initialized geoscience predictions. However, we didn't find any one package that unified all our needs.

Output from decadal climate prediction experiments is difficult to work with. A typical output file could contain the dimensions `initialization`, `lead time`, `ensemble member`, `latitude`, `longitude`, `depth`. `climpred` leverages the labeled dimensions of `xarray` to handle the headache of bookkeeping for you. We offer *`HindcastEnsemble`* and *`PerfectModelEnsemble`* objects that carry references (e.g., control runs, reconstructions, uninitialized ensembles) along with your decadal prediction output.

When computing lead-dependent skill scores, `climpred` handles all of the lag-correlating for you. We offer a suite of vectorized deterministic and probabilistic metrics that can be applied to time series and grids. It's as easy as adding your decadal prediction output to an object and running compute: `HindcastEnsemble.compute_metric(metric='rmse')`.

2.2 Scope of climpred

climpred aims to be the primary package used to analyze output from initialized dynamical forecast models, ranging from short-term weather forecasts to decadal climate forecasts. The code base will be driven entirely by the geoscientific prediction community through open source development. It leverages xarray to keep track of core prediction ensemble dimensions (e.g., ensemble member, initialization date, and lead time) and dask to perform out-of-memory computations on large datasets.

The primary goal of climpred is to offer a comprehensive set of analysis tools for assessing the forecasts relative to references (e.g., observations, reanalysis products, control runs, baseline forecasts). This will range from simple deterministic and probabilistic verification metrics—such as mean absolute error and various skill scores—to more advanced analysis methods, such as relative entropy and mutual information. climpred expects users to handle their domain-specific post-processing of model output, so that the package can focus on the actual analysis of forecasts.

Finally, the climpred documentation will serve as a repository of unified analysis methods through jupyter notebook examples, and will also collect relevant references and literature.

2.3 Quick Start

The easiest way to get up and running is to load in one of our example datasets (or load in some data of your own) and to convert them to either a HindcastEnsemble or PerfectModelEnsemble object.

climpred provides example datasets from the MPI-ESM-LR decadal prediction ensemble and the CESM decadal prediction ensemble. See our examples to see some analysis cases.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import xarray as xr

from climpred import HindcastEnsemble
import climpred
```

You can view the datasets available to be loaded with the load_datasets() command without passing any arguments:

[2]:
```
climpred.tutorial.load_dataset()
```

```
'MPI-control-1D': area averages for the MPI control run of SST/SSS.
'MPI-control-3D': lat/lon/time for the MPI control run of SST/SSS.
'MPI-PM-DP-1D': perfect model decadal prediction ensemble area averages of SST/SSS/
↪AMO.
'MPI-PM-DP-3D': perfect model decadal prediction ensemble lat/lon/time of SST/SSS/AMO.
'CESM-DP-SST': hindcast decadal prediction ensemble of global mean SSTs.
'CESM-DP-SSS': hindcast decadal prediction ensemble of global mean SSS.
'CESM-DP-SST-3D': hindcast decadal prediction ensemble of eastern Pacific SSTs.
'CESM-LE': uninitialized ensemble of global mean SSTs.
'MPIESM_miklip_baseline1-hind-SST-global': hindcast initialized ensemble of global␣
↪mean SSTs
'MPIESM_miklip_baseline1-hist-SST-global': uninitialized ensemble of global mean SSTs
'MPIESM_miklip_baseline1-assim-SST-global': assimilation in MPI-ESM of global mean␣
↪SSTs
'ERSST': observations of global mean SSTs.
'FOSI-SST': reconstruction of global mean SSTs.
'FOSI-SSS': reconstruction of global mean SSS.
'FOSI-SST-3D': reconstruction of eastern Pacific SSTs
```

From here, loading a dataset is easy. Note that you need to be connected to the internet for this to work – the datasets are being pulled from the climpred-data repository. Once loaded, it is cached on your computer so you can reload extremely quickly. These datasets are very small (< 1MB each) so they won't take up much space.

[3]:
```
hind = climpred.tutorial.load_dataset('CESM-DP-SST')
obs = climpred.tutorial.load_dataset('ERSST')
```

Make sure your prediction ensemble's dimension labeling conforms to `climpred`'s standards. In other words, you need an `init`, `lead`, and (optional) `member` dimension. Make sure that your `init` and `lead` dimensions align. *E.g.*, a November 1st, 1954 initialization should be labeled as `init=1954` so that the lead=1 forecast is 1955.

[4]:
```
print(hind)
```

```
<xarray.Dataset>
Dimensions:  (init: 64, lead: 10, member: 10)
Coordinates:
  * lead     (lead) int32 1 2 3 4 5 6 7 8 9 10
  * member   (member) int32 1 2 3 4 5 6 7 8 9 10
  * init     (init) float32 1954.0 1955.0 1956.0 1957.0 ... 2015.0 2016.0 2017.0
Data variables:
    SST      (init, lead, member) float64 ...
```

We'll quickly process the data to create anomalies. CESM-DPLE's drift-correction occurs over 1964-2014, so we'll remove that from the observations.

[5]:
```
# subtract climatology
obs = obs - obs.sel(time=slice(1964, 2014)).mean()
```

We can now create a HindcastEnsemble object and add our reference and name it `'Obs'`.

[6]:
```
hindcast = HindcastEnsemble(hind)
hindcast = hindcast.add_reference(obs, 'Obs')
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 ...
Obs:
    SST      (time) float32 -0.40146065 -0.35238647 ... 0.34601402 0.45021248
Uninitialized:
    None
```

We'll remove a linear trend so that it doesn't artificially boost our predictability. Note that `climpred` objects (HindcastEnsemble and PerfectModelEnsemble) can have any arbitrary `xarray` function applied to them. Here, we use the `xarray .apply()` function to apply our `climpred` trend removal function.

[7]:
```
# Apply the `rm_trend` function twice to detrend our obs over time and
# detrend our initialized forecasts over init. The objects ignore an xarray
# operation if the dimension doesn't exist for the given dataset.
hindcast = hindcast.apply(climpred.stats.rm_trend, dim='time')
hindcast = hindcast.apply(climpred.stats.rm_trend, dim='init')
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 0.005165 0.03014 ... 0.1842 0.1812
Obs:
    SST      (time) float32 -0.061960407 -0.023283795 ... 0.072058104 0.165859
```

(continues on next page)

(continued from previous page)

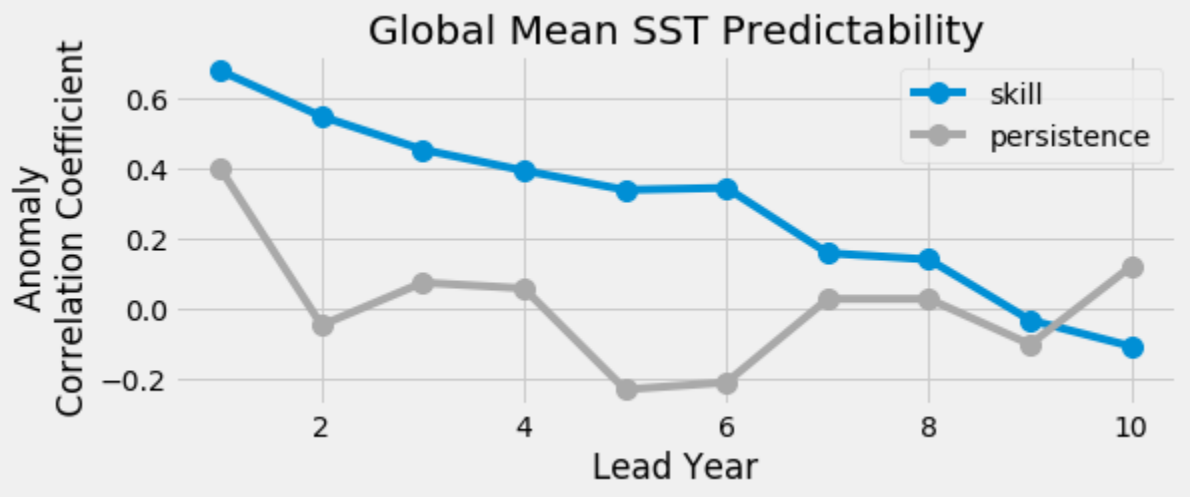
```
Uninitialized:
    None
```

Now we'll quickly calculate skill and persistence. We have a variety of possible metrics to use.

[8]:
```
init = hindcast.compute_metric(metric='acc')
persistence = hindcast.compute_persistence(metric='acc')
print(init)
```

```
<xarray.Dataset>
Dimensions:  (lead: 10)
Coordinates:
  * lead     (lead) int64 1 2 3 4 5 6 7 8 9 10
Data variables:
    SST      (lead) float64 0.6778 0.5476 0.4527 ... 0.1393 -0.03366 -0.1084
Attributes:
    prediction_skill:              calculated by climpred https://climpred.re...
    skill_calculated_by_function:  compute_hindcast
    number_of_initializations:     64
    number_of_members:             10
    metric:                        pearson_r
    comparison:                    e2r
    units:                         None
    created:                       2020-01-07 19:04:38
```

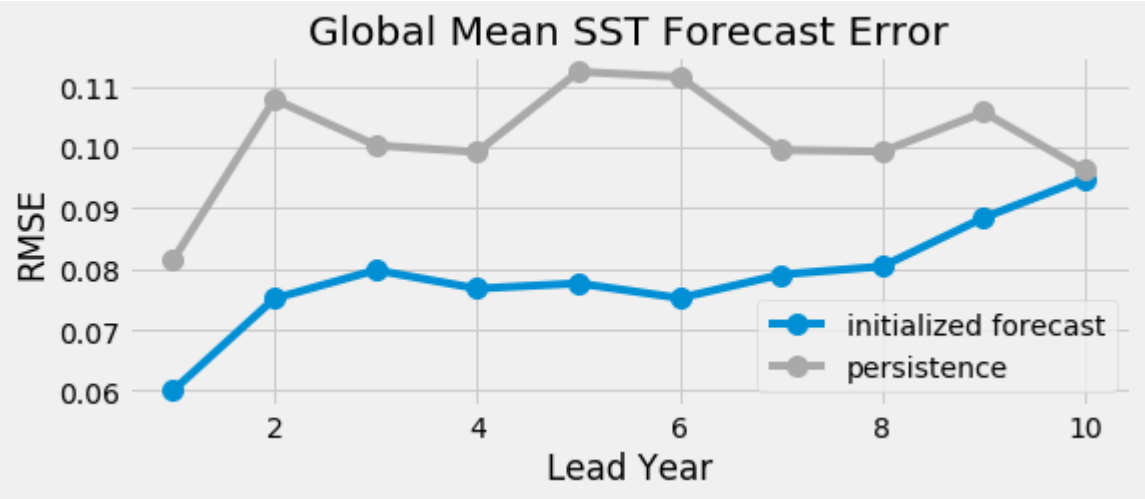
[9]:
```
plt.style.use('fivethirtyeight')
f, ax = plt.subplots(figsize=(8, 3))
init.SST.plot(marker='o', markersize=10, label='skill')
persistence.SST.plot(marker='o', markersize=10, label='persistence',
                     color='#a9a9a9')
plt.legend()
ax.set(title='Global Mean SST Predictability',
       ylabel='Anomaly \n Correlation Coefficient',
       xlabel='Lead Year')
plt.show()
```



We can also check error in our forecasts.

[10]:
```
init = hindcast.compute_metric(metric='rmse')
persistence = hindcast.compute_persistence(metric='rmse')
```

[11]:
```
plt.style.use('fivethirtyeight')
f, ax = plt.subplots(figsize=(8, 3))
init.SST.plot(marker='o', markersize=10, label='initialized forecast')
persistence.SST.plot(marker='o', markersize=10, label='persistence',
                     color='#a9a9a9')
plt.legend()
ax.set(title='Global Mean SST Forecast Error',
       ylabel='RMSE',
       xlabel='Lead Year')
plt.show()
```



2.4 Examples

2.4.1 Demo of Perfect Model Predictability Functions

This demo demonstrates `climpred`'s capabilities for a perfect-model framework ensemble simulation.

What's a perfect-model framework simulation?

A perfect-model framework uses a set of ensemble simulations that are based on a General Circulation Model (GCM) or Earth System Model (ESM) alone. There is *no* use of any reanalysis, reconstruction, or data product to initialize the decadal prediction ensemble. An arbitrary number of `members` are initialized from perturbed initial conditions (the "`ensemble`"), and the control simulation can be viewed as just another member.

How to compare predictability skill score: As no observational data interferes with the random climate evolution of the model, we cannot use an observation-based reference for computing skill scores. Therefore, we can compare the members with one another (`m2m`), against the ensemble mean (`m2e`), or against the control (`m2c`). We can also compare the ensemble mean to the control (`e2c`). See the comparisons page for more information.

When to use perfect-model frameworks:

- You don't have a sufficiently long observational record to use as a reference.
- You want to avoid biases between model climatology and reanalysis climatology.

- You want to avoid sensitive reactions of biogeochemical cycles to disruptive changes in ocean physics due to assimilation.
- You want to delve into process understanding of predictability in a model without outside artifacts.

[1]:
```
import warnings

import cartopy.crs as ccrs
import cartopy.feature as cfeature
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import xarray as xr

import climpred
```

[2]:
```
warnings.filterwarnings("ignore")
```

Load sample data

Here we use a subset of ensembles and members from the MPI-ESM-LR (CMIP6 version) esmControl simulation of an early state. This corresponds to `vga0214` from year 3000 to 3300.

1-dimensional output

Our 1D sample output contains datasets of time series of certain spatially averaged `area` ('global', 'North_Atlantic') and temporally averaged `period` ('ym', 'DJF', ...) for some `lead` years (1, ..., 20).

`ds`: The ensemble dataset of all `members` (1, ..., 10), `inits` (initialization years: 3014, 3023, ..., 3257), `areas`, `periods`, and `lead` years.

`control`: The control dataset with the same areas and periods, as well as the years 3000 to 3299.

[3]:
```
ds = climpred.tutorial.load_dataset('MPI-PM-DP-1D')
control = climpred.tutorial.load_dataset('MPI-control-1D')
```

[4]:
```
# Add to climpred PerfectModelEnsemble object.
pm = climpred.PerfectModelEnsemble(ds)
pm = pm.add_control(control)
print(pm)
```

```
<climpred.PerfectModelEnsemble>
Initialized Ensemble:
    tos      (period, lead, area, init, member) float32 ...
    sos      (period, lead, area, init, member) float32 ...
    AMO      (period, lead, area, init, member) float32 ...
Control:
    tos      (period, time, area) float32 ...
    sos      (period, time, area) float32 ...
    AMO      (period, time, area) float32 ...
Uninitialized:
    None
```

We'll sub-select annual means ('ym') of sea surface temperature ('tos') in the North Atlantic.

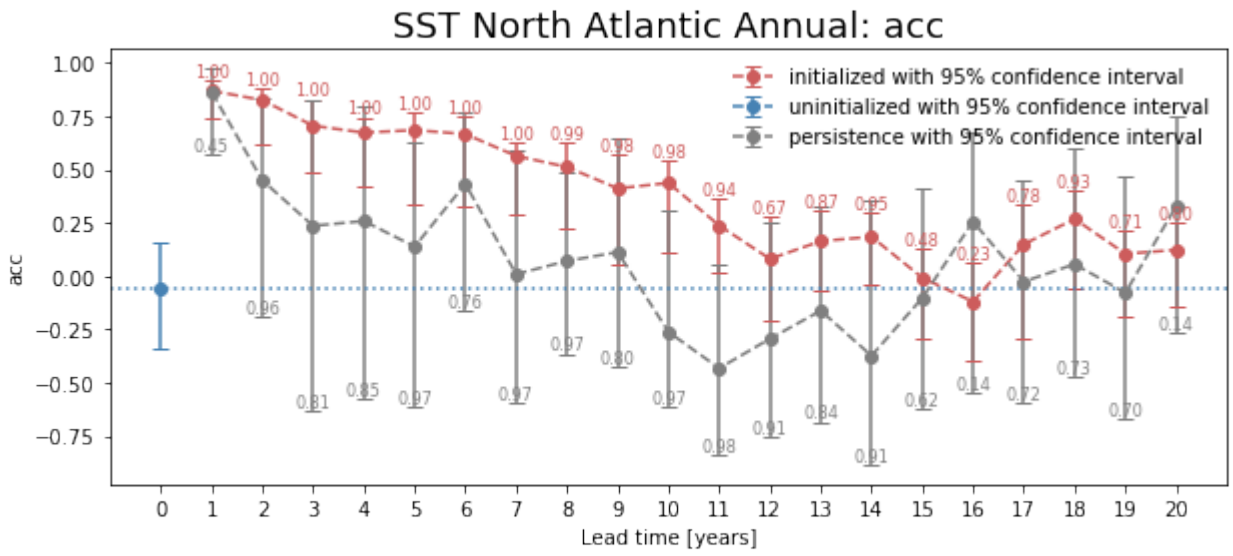
[5]:
```
# Currently cannot sub-select variables. Easiest way is to just use drop, or if there
↪'s lots
# of variables, select them before creating the object.
pm = pm.sel(area='North_Atlantic', period='ym').drop(['sos', 'AMO']).reset_
↪coords(drop=True)
```

Bootstrapping with Replacement

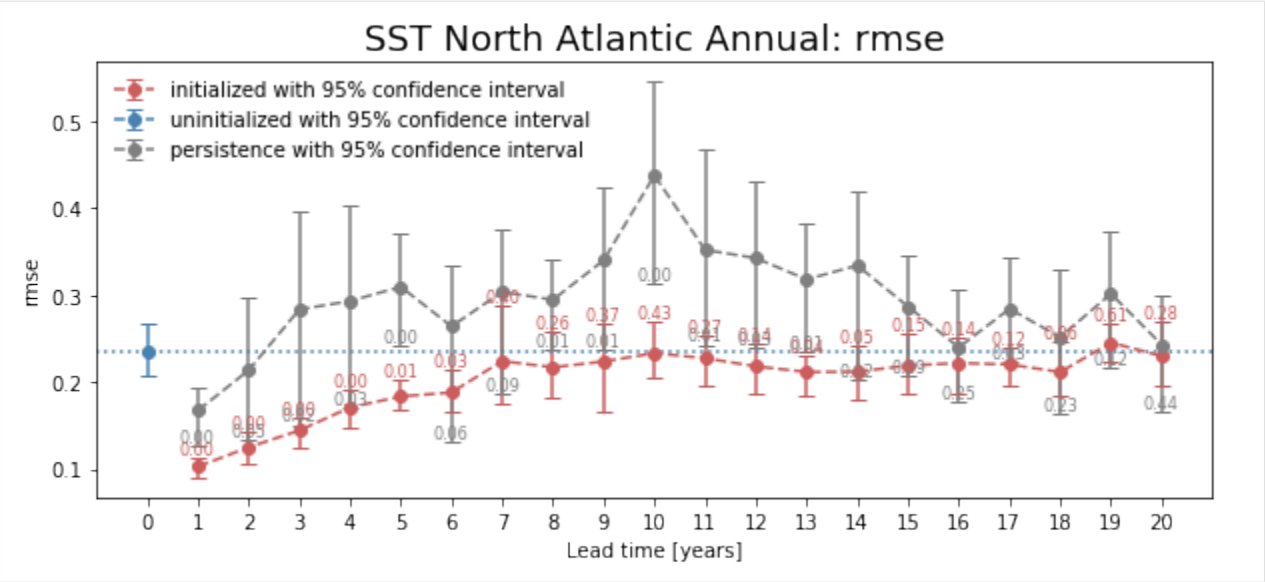
Here, we bootstrap the ensemble with replacement [Goddard et al. 2013] to compare the initialized ensemble to an "uninitialized" counterpart and a persistence forecast. The visualization is based on those used in [Li et al. 2016]. The p value demonstrates the probability that the uninitialized or persistence beats the initialized forecast based on N=100 bootstrapping with replacement.

[6]:
```
for metric in ['acc', 'rmse']:
    bootstrapped = pm.bootstrap(metric=metric, comparison='m2e', bootstrap=100,␣
↪sig=95)
    # Hacky fix that needs to be dealt with in a PR.
    # climpred objects return a dataset. graphics module wants a DataArray but looks
    # for the attributes that are attached to the Dataset.
    bs = bootstrapped['tos']
    bs.attrs = bootstrapped.attrs
    climpred.graphics.plot_bootstrapped_skill_over_leadyear(bs, sig=95)
    plt.title(' '.join(['SST', 'North Atlantic', 'Annual:', metric]),fontsize=18)
    plt.ylabel(metric)
    plt.show()
```

```
bootstrapping iteration: 100%|| 100/100 [00:39<00:00,  2.55it/s]
```



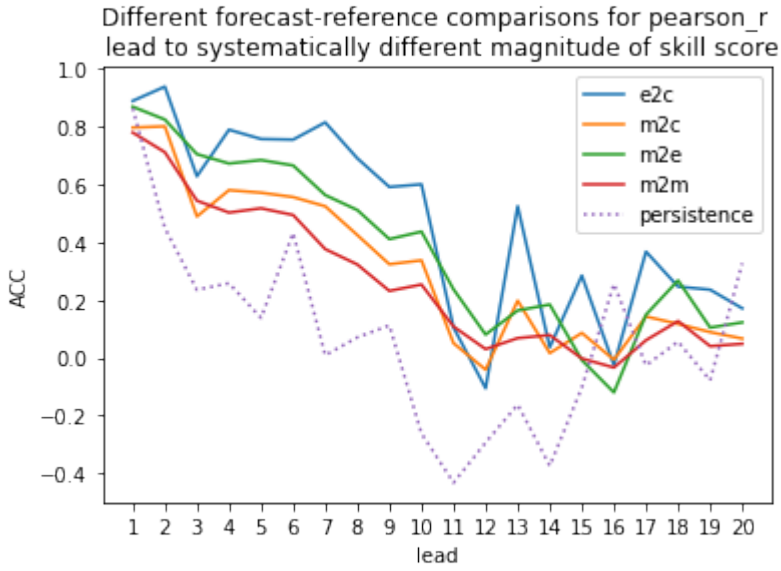

```
bootstrapping iteration: 100%|| 100/100 [00:37<00:00,  2.69it/s]
```



Computing Skill with Different Comparison Methods

Here, we use `compute_perfect_model` to compute the Anomaly Correlation Coefficient (ACC) with different comparison methods. This generates different ACC values by design. See the comparisons page for a description of the various ways to compute skill scores for a perfect-model framework.

[7]:
```
for c in ['e2c','m2c','m2e','m2m']:
    pm.compute_metric(metric='acc', comparison=c)['tos'].plot(label=c)
# Persistence computation for a baseline.
pm.compute_persistence(metric='acc')['tos'].plot(label='persistence', ls=':')
plt.ylabel('ACC')
plt.xticks(np.arange(1,21))
plt.legend()
plt.title('Different forecast-reference comparisons for pearson_r \n lead to␣
↪systematically different magnitude of skill score')
plt.show()
```



3-dimensional output (maps)

We also have some sample output that contains gridded time series on the curvilinear MPI grid. Our compute functions (`compute_perfect_model`, `compute_persistence`) are indifferent to any dimensions that exist in addition to `init`, `member`, and `lead`. In other words, the functions are set up to make these computations on a grid, if one includes `lat`, `lon`, `lev`, `depth`, etc.

`ds3d`: The ensemble dataset of `members` (1, 2, 3, 4), `inits` (initialization years: 3014, 3061, 3175, 3237), and `lead` years (1, 2, 3, 4, 5).

`control3d`: The control dataset spanning (3000, . . . , 3049).

Note: These are very small subsets of the actual MPI simulations so that we could host the sample output maps on Github.

[8]:
```
# Sea surface temperature
ds3d = climpred.tutorial.load_dataset('MPI-PM-DP-3D') \
                        .sel(init=3014) \
                        .expand_dims('init')['tos']
control3d = climpred.tutorial.load_dataset('MPI-control-3D')['tos']
```

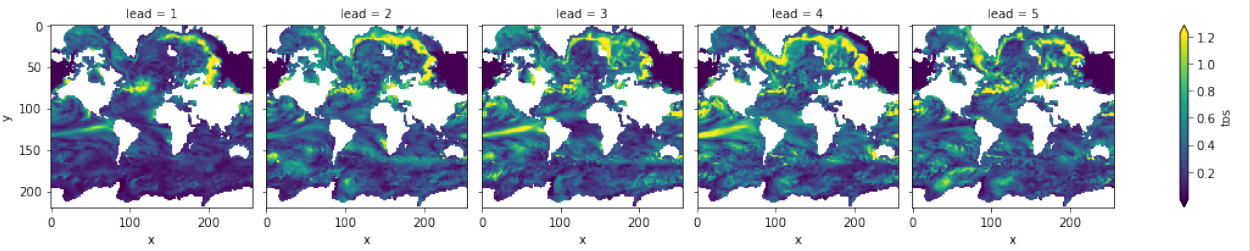
[9]:
```
# Create climpred PerfectModelEnsemble object.
pm = climpred.PerfectModelEnsemble(ds3d)
pm = pm.add_control(control3d)
print(pm)
```

```
<climpred.PerfectModelEnsemble>
Initialized Ensemble:
    tos      (init, lead, member, y, x) float32 nan nan nan nan ... nan nan nan
Control:
    tos      (time, y, x) float32 ...
Uninitialized:
    None
```

Maps of Skill by Lead Year

[10]:
```
pm.compute_metric(metric='rmse', comparison='m2e')['tos'].T.plot(col='lead',␣
↪robust=True, yincrease=False)
```

[10]:
```
<xarray.plot.facetgrid.FacetGrid at 0x11abb3cf8>
```



References

1. Bushuk, Mitchell, Rym Msadek, Michael Winton, Gabriel Vecchi, Xiaosong Yang, Anthony Rosati, and Rich Gudgel. "Regional Arctic Sea–Ice Prediction: Potential versus Operational Seasonal Forecast Skill." Climate Dynamics, June 9, 2018. https://doi.org/10/gd7hfq.
2. Collins, Matthew, and Sinha Bablu. "Predictability of Decadal Variations in the Thermohaline Circulation and Climate." Geophysical Research Letters 30, no. 6 (March 22, 2003). https://doi.org/10/cts3cr.
3. Goddard, Lisa, et al. "A verification framework for interannual-to-decadal predictions experiments." Climate Dynamics 40.1-2 (2013): 245-272.
4. Griffies, S. M., and K. Bryan. "A Predictability Study of Simulated North Atlantic Multidecadal Variability." Climate Dynamics 13, no. 7–8 (August 1, 1997): 459–87. https://doi.org/10/ch4kc4.
5. Hawkins, Ed, Steffen Tietsche, Jonathan J. Day, Nathanael Melia, Keith Haines, and Sarah Keeley. "Aspects of Designing and Evaluating Seasonal-to-Interannual Arctic Sea-Ice Prediction Systems." Quarterly Journal of the Royal Meteorological Society 142, no. 695 (January 1, 2016): 672–83. https://doi.org/10/gfb3pn.
6. Li, Hongmei, Tatiana Ilyina, Wolfgang A. Müller, and Frank Sienz. "Decadal Predictions of the North Atlantic CO2 Uptake." Nature Communications 7 (March 30, 2016): 11076. https://doi.org/10/f8wkrs.
7. Pohlmann, Holger, Michael Botzet, Mojib Latif, Andreas Roesch, Martin Wild, and Peter Tschuck. "Estimating the Decadal Predictability of a Coupled AOGCM." Journal of Climate 17, no. 22 (November 1, 2004): 4463–72. https://doi.org/10/d2qf62.

2.4.2 Hindcast Predictions of Equatorial Pacific SSTs

In this example, we evaluate hindcasts (retrospective forecasts) of sea surface temperatures in the eastern equatorial Pacific from CESM-DPLE. These hindcasts are evaluated against a forced ocean–sea ice simulation that initializes the model.

See the quick start for an analysis of time series (rather than maps) from a hindcast prediction ensemble.

[1]:
```
import warnings

import cartopy.crs as ccrs
import cartopy.feature as cfeature
%matplotlib inline
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import numpy as np

import climpred
from climpred import HindcastEnsemble
```

[2]:
```
warnings.filterwarnings("ignore")
```

We'll load in a small region of the eastern equatorial Pacific for this analysis example.

[3]:
```
climpred.tutorial.load_dataset()
```

```
'MPI-control-1D': area averages for the MPI control run of SST/SSS.
'MPI-control-3D': lat/lon/time for the MPI control run of SST/SSS.
'MPI-PM-DP-1D': perfect model decadal prediction ensemble area averages of SST/SSS/
↪AMO.
'MPI-PM-DP-3D': perfect model decadal prediction ensemble lat/lon/time of SST/SSS/AMO.
'CESM-DP-SST': hindcast decadal prediction ensemble of global mean SSTs.
'CESM-DP-SSS': hindcast decadal prediction ensemble of global mean SSS.
'CESM-DP-SST-3D': hindcast decadal prediction ensemble of eastern Pacific SSTs.
'CESM-LE': uninitialized ensemble of global mean SSTs.
'MPIESM_miklip_baseline1-hind-SST-global': hindcast initialized ensemble of global␣
↪mean SSTs
'MPIESM_miklip_baseline1-hist-SST-global': uninitialized ensemble of global mean SSTs
'MPIESM_miklip_baseline1-assim-SST-global': assimilation in MPI-ESM of global mean␣
↪SSTs
'ERSST': observations of global mean SSTs.
'FOSI-SST': reconstruction of global mean SSTs.
'FOSI-SSS': reconstruction of global mean SSS.
'FOSI-SST-3D': reconstruction of eastern Pacific SSTs
```

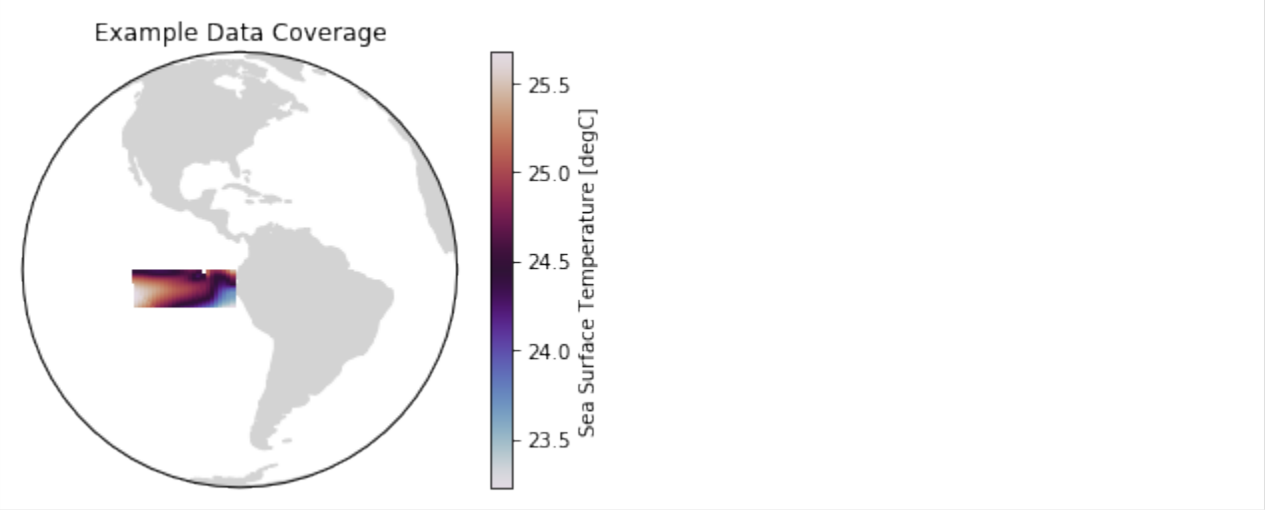
[4]:
```
hind = climpred.tutorial.load_dataset('CESM-DP-SST-3D')['SST']
recon = climpred.tutorial.load_dataset('FOSI-SST-3D')['SST']
print(hind)
```

```
<xarray.DataArray 'SST' (init: 64, lead: 10, nlat: 37, nlon: 26)>
[615680 values with dtype=float32]
Coordinates:
    TLAT     (nlat, nlon) float64 ...
    TLONG    (nlat, nlon) float64 ...
  * init     (init) float32 1954.0 1955.0 1956.0 1957.0 ... 2015.0 2016.0 2017.0
  * lead     (lead) int32 1 2 3 4 5 6 7 8 9 10
    TAREA    (nlat, nlon) float64 ...
Dimensions without coordinates: nlat, nlon
```

These two example products cover a small portion of the eastern equatorial Pacific.

[5]:
```
ax = plt.axes(projection=ccrs.Orthographic(-80, 0))
p = ax.pcolormesh(recon.TLONG, recon.TLAT, recon.mean('time'),
                  transform=ccrs.PlateCarree(), cmap='twilight')
ax.add_feature(cfeature.LAND, color='#d3d3d3')
ax.set_global()
plt.colorbar(p, label='Sea Surface Temperature [degC]')
ax.set(title='Example Data Coverage')
```

[5]:
```
[Text(0.5, 1.0, 'Example Data Coverage')]
```



We first need to remove the same climatology that was used to drift-correct the CESM-DPLE. Then we'll create a detrended version of our two products to assess detrended predictability.

[6]:
```
# Remove 1964-2014 climatology.
recon = recon - recon.sel(time=slice(1964, 2014)).mean('time')
```

Although functions can be called directly in `climpred`, we suggest that you use our classes (`HindcastEnsemble` and `PerfectModelEnsemble`) to make analysis code cleaner.

[7]:
```
hindcast = HindcastEnsemble(hind)
hindcast = hindcast.add_reference(recon, 'Reconstruction')
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, nlat, nlon) float32 ...
Reconstruction:
    SST      (time, nlat, nlon) float32 0.0029411316 0.0013866425 ... 1.4646168
Uninitialized:
    None
```

I'll also detrend the reconstruction over its `time` dimension and initialized forecast ensemble over `init`.

[8]:
```
# Apply the `rm_trend` function twice to detrend our obs over time and
# detrend our initialized forecasts over init. The objects ignore an xarray
# operation if the dimension doesn't exist for the given dataset.
hindcast = hindcast.apply(climpred.stats.rm_trend, dim='init')
hindcast = hindcast.apply(climpred.stats.rm_trend, dim='time')
```

Anomaly Correlation Coefficient of SSTs

We can now compute the ACC over all leads and all grid cells.

[9]:
```
predictability = hindcast.compute_metric(metric='acc')
print(predictability)
```

```
<xarray.Dataset>
Dimensions:  (lead: 10, nlat: 37, nlon: 26)
```

(continues on next page)

(continued from previous page)

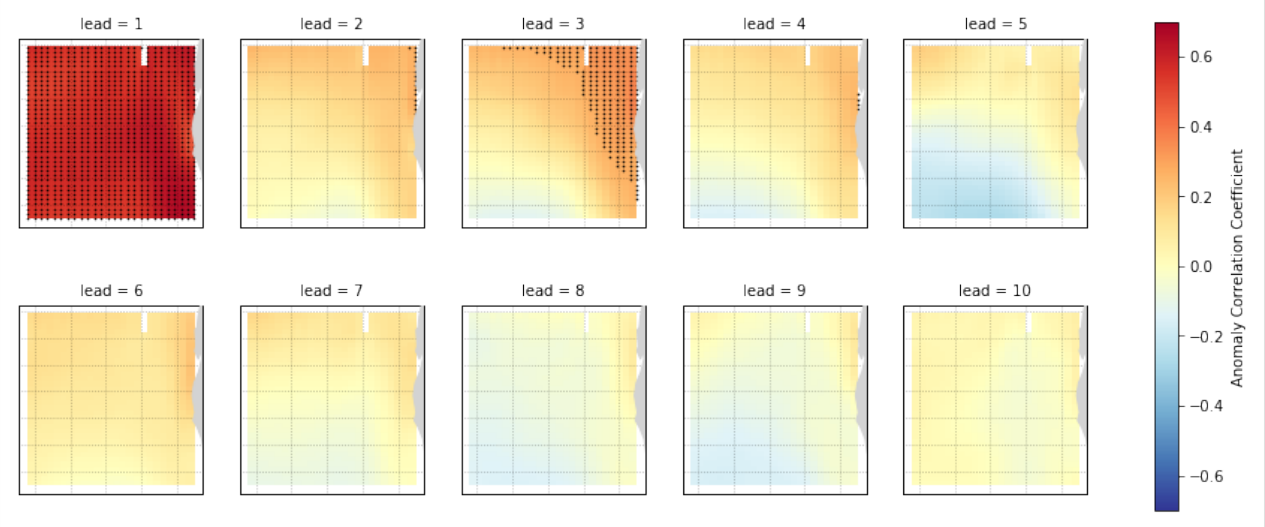
```
Coordinates:
    TLONG    (lead, nlat, nlon) float64 250.8 251.9 253.1 ... 276.7 277.8 278.9
    TAREA    (lead, nlat, nlon) float64 3.661e+13 3.661e+13 ... 3.714e+13
  * nlat     (nlat) int64 0 1 2 3 4 5 6 7 8 9 ... 27 28 29 30 31 32 33 34 35 36
  * nlon     (nlon) int64 0 1 2 3 4 5 6 7 8 9 ... 16 17 18 19 20 21 22 23 24 25
  * lead     (lead) int64 1 2 3 4 5 6 7 8 9 10
    TLAT     (lead, nlat, nlon) float64 -9.75 -9.75 -9.75 ... -0.1336 -0.1336
Data variables:
    SST      (lead, nlat, nlon) float32 0.54588705 0.53977644 ... 0.088374
Attributes:
    prediction_skill:              calculated by climpred https://climpred.re...
    skill_calculated_by_function:  compute_hindcast
    number_of_initializations:     64
    metric:                        pearson_r
    comparison:                    e2r
    units:                         None
    created:                       2019-11-21 15:52:28
```

We use the `pval` keyword to get associated p values for our ACCs. We can then mask our final maps based on $\alpha = 0.05$.

[10]:
```
significance = hindcast.compute_metric(metric='p_pval')

# Mask latitude and longitude by significance for stippling.
siglat = significance.TLAT.where(significance.SST <= 0.05)
siglon = significance.TLONG.where(significance.SST <= 0.05)
```

[11]:
```
p = predictability.SST.plot.pcolormesh(x='TLONG', y='TLAT',
                                       transform=ccrs.PlateCarree(),
                                       col='lead', col_wrap=5,
                                       subplot_kws={'projection': ccrs.PlateCarree(),
                                                    'aspect': 3},
                                       cbar_kwargs={'label': 'Anomaly Correlation␣
↪Coefficient'},
                                       vmin=-0.7, vmax=0.7,
                                       cmap='RdYlBu_r')
for i, ax in enumerate(p.axes.flat):
    ax.add_feature(cfeature.LAND, color='#d3d3d3', zorder=4)
    ax.gridlines(alpha=0.3, color='k', linestyle=':')
    # Add significance stippling
    ax.scatter(siglon.isel(lead=i),
               siglat.isel(lead=i),
               color='k',
               marker='.',
               s=1.5,
               transform=ccrs.PlateCarree())
```

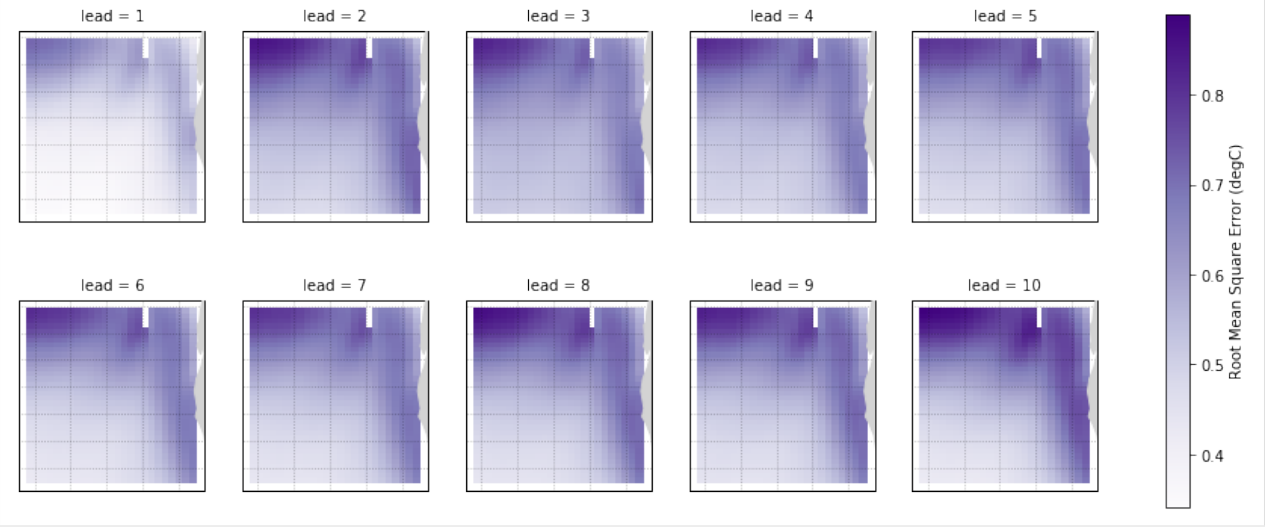


Root Mean Square Error of SSTs

We can also check error in our forecasts, just by changing the metric keyword.

[12]:
```
rmse = hindcast.compute_metric(metric='rmse')
```

[13]:
```
p = rmse.SST.plot.pcolormesh(x='TLONG', y='TLAT',
                             transform=ccrs.PlateCarree(),
                             col='lead', col_wrap=5,
                             subplot_kws={'projection': ccrs.PlateCarree(),
                                          'aspect': 3},
                             cbar_kwargs={'label': 'Root Mean Square Error (degC)'},
                             cmap='Purples')
for ax in p.axes.flat:
    ax.add_feature(cfeature.LAND, color='#d3d3d3', zorder=4)
    ax.gridlines(alpha=0.3, color='k', linestyle=':')
```



[ ]:

2.4.3 Diagnosing Potential Predictability

This demo demonstrates `climpred`'s capabilities to diagnose areas containing potentially predictable variations from a control or reference alone without requiring multi-member, multi-initialization simulations. This notebook identifies the slow components of internal variability that indicate potential predictability. Here, we showcase a set of methods to show regions indicating probabilities for decadal predictability.

[1]:
```
import warnings
%matplotlib inline
import climpred
warnings.filterwarnings("ignore")
```

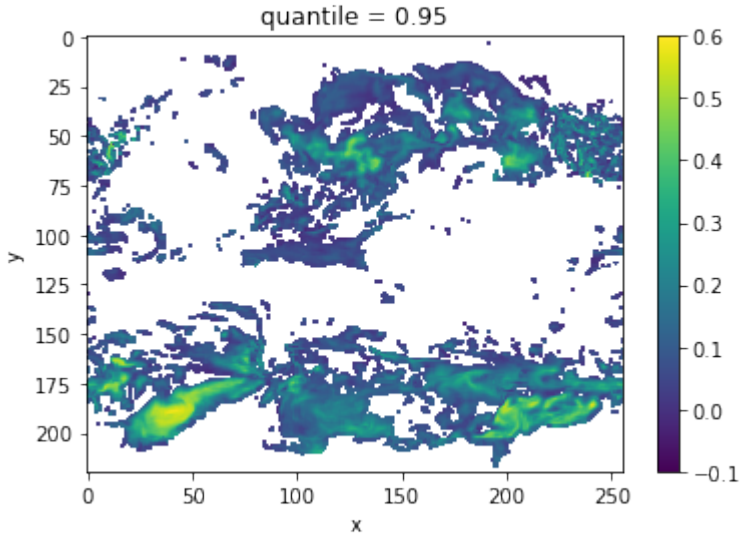
[2]:
```
# Sea surface temperature
varname='tos'
control3d = climpred.tutorial.load_dataset('MPI-control-3D')[varname].load()
```

Diagnostic Potential Predictability (DPP)

We can first use the [Resplandy 2015] and [Seferian 2018] method for computing the unbiased DPP by not chunking the time dimension.

[3]:
```
# calculate DPP with m=10
DPP10 = climpred.stats.dpp(control3d, m=10, chunk=False)
# calculate a threshold by random shuffling (based on bootstrapping with replacement␣
↪at 95% significance level)
threshold = climpred.bootstrap.dpp_threshold(control3d,
                                             m=10,
                                             chunk=False,
                                             bootstrap=10,
                                             sig=95)
# plot grid cells where DPP above threshold
DPP10.where(DPP10 > threshold).plot(yincrease=False, vmin=-0.1, vmax=0.6, cmap=
↪'viridis')
```

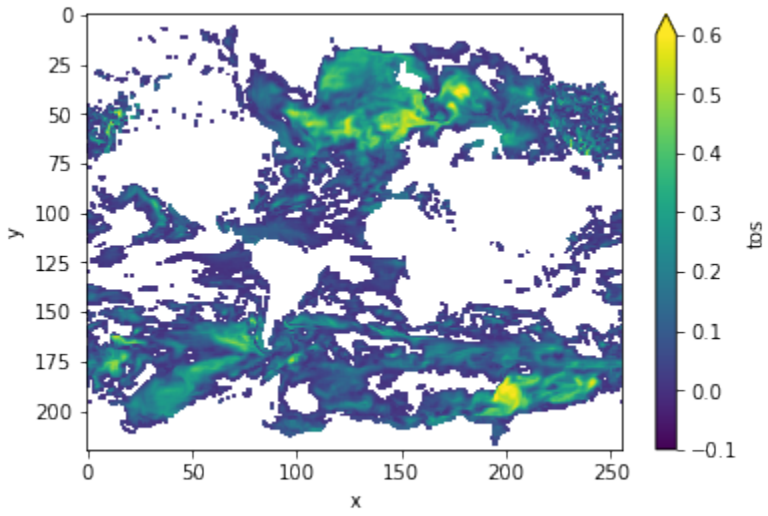
[3]:
```
<matplotlib.collections.QuadMesh at 0x7fa3315bb908>
```



Now, we can turn on chunking (the default for this function) to use the [Boer 2004] method.

[4]:
```
# chunk = True signals the Boer 2004 method
DPP10 = climpred.stats.dpp(control3d, m=10, chunk=True)
threshold = climpred.bootstrap.dpp_threshold(control3d,
                                             m=10,
                                             chunk=True,
                                             bootstrap=50,
                                             sig=95)
DPP10.where(DPP10>0).plot(yincrease=False, vmin=-0.1, vmax=0.6, cmap='viridis')
```

[4]:
```
<matplotlib.collections.QuadMesh at 0x7fa331c7e0b8>
```

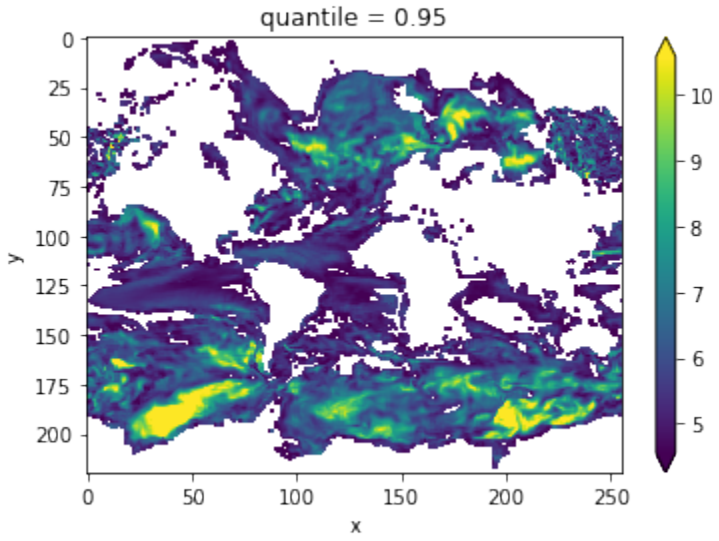


Variance-Weighted Mean Period

A periodogram is computed based on a control simulation to extract the mean period of variations, which are weighted by the respective variance. Regions with a high mean period value indicate low-frequency variations with are potentially predictable [Branstator 2010].

[5]:
```
vwmp = climpred.stats.varweighted_mean_period(control3d, dim='time')
threshold = climpred.bootstrap.varweighted_mean_period_threshold(control3d,
                                                                 bootstrap=10)
vwmp.where(vwmp > threshold).plot(yincrease=False, robust=True)
```

[5]:
```
<matplotlib.collections.QuadMesh at 0x7fa3131b7240>
```

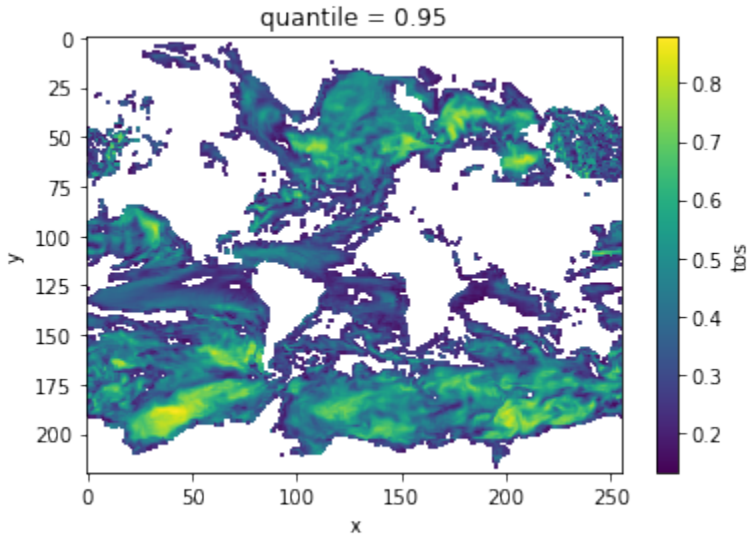


Lag-1 Autocorrelation

The lag-1 autocorrelation also indicates where slower modes of variability occur by identifying regions with high temporal correlation [vonStorch 1999].

[6]:
```
# use climpred.bootstrap._bootstrap_func to wrap any stats function
threshold = climpred.bootstrap._bootstrap_func(climpred.stats.autocorr,control3d,'time
↪',bootstrap=100)
corr_ef = climpred.stats.autocorr(control3d, dim='time')
corr_ef.where(corr_ef>threshold).plot(yincrease=False, robust=False)
```

[6]:
```
<matplotlib.collections.QuadMesh at 0x7fa3213b8cc0>
```

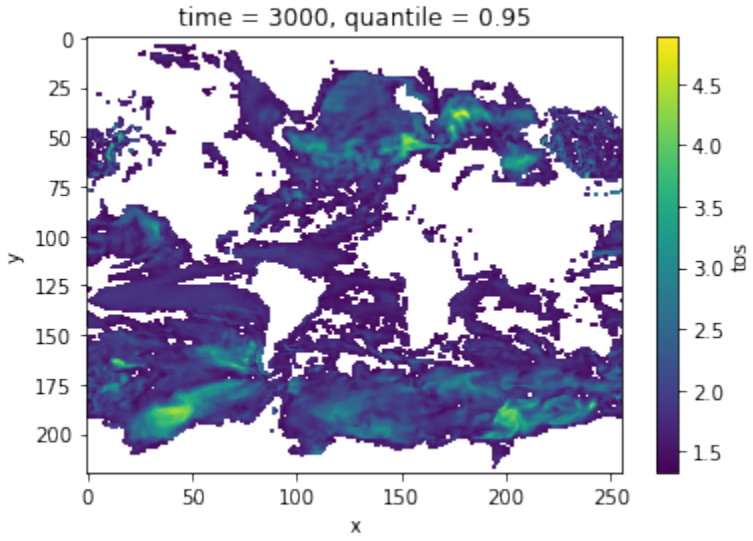


Decorrelation time

Taking the lagged correlation further over all lags, the decorrelation time shows the time after which the autocorrelation fell beyond its e-folding [vonStorch 1999]

[7]:
```
threshold = climpred.bootstrap._bootstrap_func(climpred.stats.decorrelation_time,
↪control3d,'time',bootstrap=100)
decorr_time = climpred.stats.decorrelation_time(control3d)
decorr_time.where(decorr_time>threshold).plot(yincrease=False, robust=False)
```

[7]:
```
<matplotlib.collections.QuadMesh at 0x7fa3312afa58>
```



Verify diagnostic potential predictability in predictability simulations

Do we find predictability in the areas highlighted above also in perfect-model experiments?

[8]:
```
ds3d = climpred.tutorial.load_dataset('MPI-PM-DP-3D')[varname].load()
```

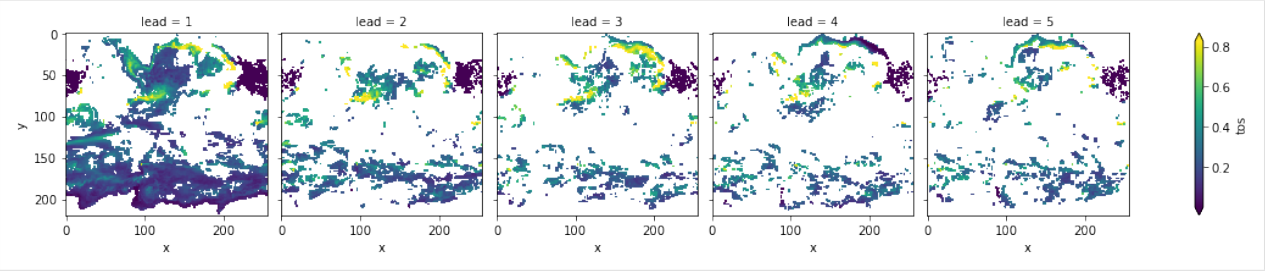
[9]:
```
bootstrap_skill = climpred.bootstrap.bootstrap_perfect_model(ds3d,
                                                control3d,
                                                metric='rmse',
                                                comparison='m2e',
                                                bootstrap=20)
```

```
HBox(children=(FloatProgress(value=0.0, description='bootstrapping iteration', max=20.
↪0, style=ProgressStyle(d...
```

[10]:
```
init_skill = bootstrap_skill.sel(results='skill',kind='init')
# p value: probability that random uninitialized forecasts perform better than␣
↪initialized
p = bootstrap_skill.sel(results='p',kind='uninit')
```

[11]:
```
init_skill.where(p<=.05).plot(col='lead', robust=True, yincrease=False)
```

[11]:
```
<xarray.plot.facetgrid.FacetGrid at 0x7fa311a2c898>
```



The metric `rmse` is negatively oriented, e.g. higher values show large disprepancy between members and hence less skill.

As suggested by DPP, the variance-weighted mean period and autocorrelation, also in slight perturbed initial values ensembles there is predictability in the North Atlantic, North Pacific and Southern Ocean in sea-surface temperatures.

References

1. Boer, Georges J. "Long time-scale potential predictability in an ensemble of coupled climate models." Climate dynamics 23.1 (2004): 29-44.
2. Resplandy, Laure, R. Séférian, and L. Bopp. "Natural variability of CO2 and O2 fluxes: What can we learn from centuries-long climate models simulations?." Journal of Geophysical Research: Oceans 120.1 (2015): 384-404.
3. Séférian, Roland, Sarah Berthet, and Matthieu Chevallier. "Assessing the Decadal Predictability of Land and Ocean Carbon Uptake." Geophysical Research Letters, March 15, 2018. https://doi.org/10/gdb424.
4. Branstator, Grant, and Haiyan Teng. "Two Limits of Initial-Value Decadal Predictability in a CGCM." Journal of Climate 23, no. 23 (August 27, 2010): 6292–6311. https://doi.org/10/bwq92h.
5. Storch, H. v, and Francis W. Zwiers. Statistical Analysis in Climate Research. Cambridge; New York: Cambridge University Press, 1999.

2.4.4 Temporal and spatial smoothing

This demo demonstrates `climpred`'s capabilities to postprocess decadal prediction output before skill verification. Here, we showcase a set of methods to smooth out noise in the spatial and temporal domain.

[1]:
```
import warnings
%matplotlib inline
import climpred
warnings.filterwarnings("ignore")
```

[2]:
```
# Sea surface temperature
varname='tos'
ds3d = climpred.tutorial.load_dataset('MPI-PM-DP-3D')[varname]
control3d = climpred.tutorial.load_dataset('MPI-control-3D')[varname]
```

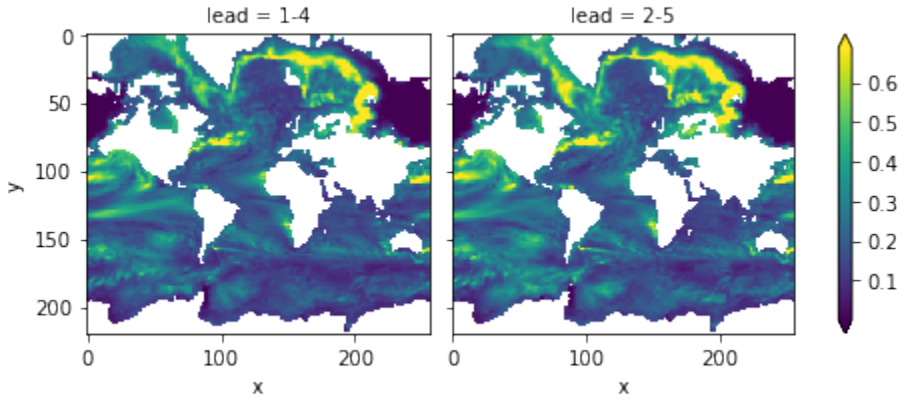
Temporal smoothing

In order to reduce temporal noise, decadal predictions are recommended to take multi-year averages [Goddard2013].

[3]:
```
ds3d_ts = climpred.smoothing.temporal_smoothing(ds3d,smooth_kws={'lead':4})
control3d_ts = climpred.smoothing.temporal_smoothing(control3d, smooth_kws={'time':4})
```

[4]:
```
climpred.prediction.compute_perfect_model(ds3d_ts,
                                          control3d_ts,
                                          metric='rmse',
                                          comparison='m2e') \
                    .plot(col='lead', robust=True, yincrease=False)
```

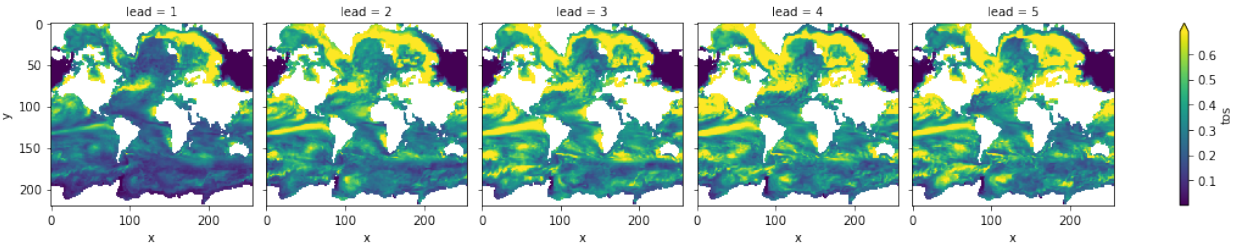
[4]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1088d1518>
```



Compare to without smoothing:

[5]:
```
climpred.prediction.compute_perfect_model(ds3d,
                                          control3d,
                                          metric='rmse',
                                          comparison='m2e') \
                    .plot(col='lead', vmax=.69, yincrease=False)
```

[5]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1249b5278>
```



Note: When using `temporal_smoothing` on compute_hindcast, set rename_dim=False and after calculating the skill `_reset_temporal_axis` to get proper labeling of the lead dimension.

[6]:
```
hind = climpred.tutorial.load_dataset('CESM-DP-SST-3D').load()['SST']
reconstruction = climpred.tutorial.load_dataset('FOSI-SST-3D').load()['SST']
# get anomaly reconstruction
reconstruction = reconstruction - reconstruction.mean('time')
```

[7]:
```
hind_ts = climpred.smoothing.temporal_smoothing(hind,smooth_kws={'lead':4},rename_
↪dim=False)
reconstruction_ts = climpred.smoothing.temporal_smoothing(reconstruction, smooth_kws={
↪'time':4},rename_dim=False)
```

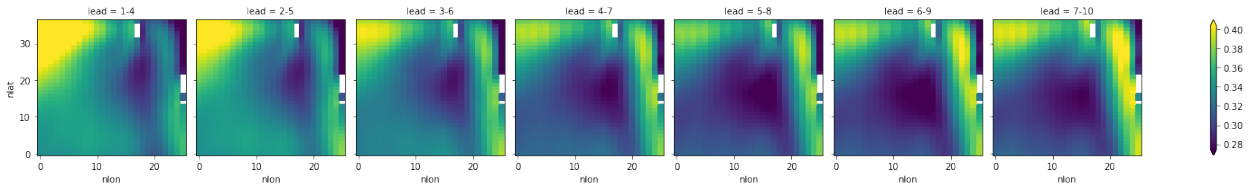
[8]:
```
s = climpred.prediction.compute_hindcast(hind_ts,
                                         reconstruction_ts,
```

(continues on next page)

(continued from previous page)

```
                                          metric='rmse',
                                          comparison='e2r')
s = climpred.smoothing._reset_temporal_axis(s,smooth_kws={'lead':4})
s.plot(col='lead', robust=True)
```

[8]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1206d6ef0>
```



Spatial smoothing

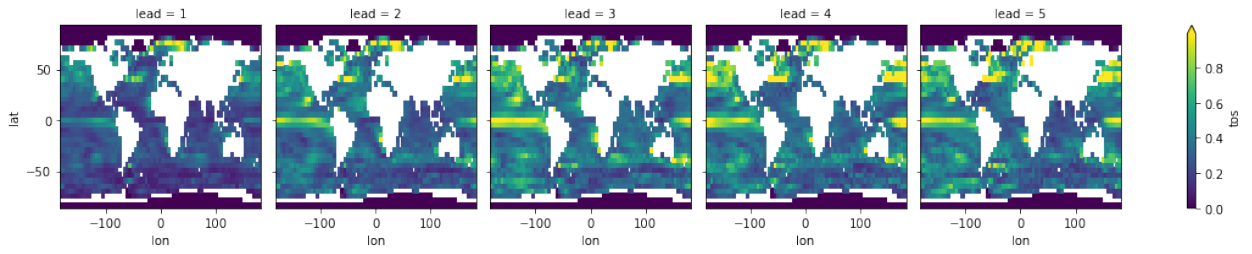
In order to reduce spatial noise, global decadal predictions are recommended to get regridded to a 5 degree longitude x 5 degree latitude grid as recommended [Goddard2013].

[9]:
```
ds3d_ss = climpred.smoothing.spatial_smoothing_xesmf(ds3d,d_lon_lat_kws={'lon':5, 'lat
↪':5})
control3d_ss = climpred.smoothing.spatial_smoothing_xesmf(control3d, d_lon_lat_kws={
↪'lon':5,'lat':5})
```

```
Create weight file: bilinear_220x256_36x73.nc
Reuse existing file: bilinear_220x256_36x73.nc
```

[10]:
```
climpred.prediction.compute_perfect_model(ds3d_ss,
                                          control3d_ss,
                                          metric='rmse',
                                          comparison='m2e') \
                    .plot(col='lead', robust=True, yincrease=True)
```

[10]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1220bf588>
```



Alternatively, also `climpred.smoothing.spatial_smoothing_xrcoarsen` aggregates gridcells like `xr_coarsen`.

`smooth_goddard2013` creates 4-year means and 5x5 degree regridding as suggested in [Goddard2013].

[11]:
```
climpred.smoothing.smooth_goddard_2013(ds3d).coords
```

```
Reuse existing file: bilinear_220x256_36x73.nc
```

[11]:
```
Coordinates:
  * lead     (lead) <U3 '1-4' '2-5'
  * init     (init) int64 3014 3061 3175 3237
  * member   (member) int64 1 2 3 4
```

(continues on next page)

(continued from previous page)

```
  * lon      (lon) float64 -180.0 -175.0 -170.0 -165.0 ... 170.0 175.0 180.0
  * lat      (lat) float64 -83.97 -78.97 -73.97 -68.97 ... 81.03 86.03 91.03
```

References

1. Goddard, L., A. Kumar, A. Solomon, D. Smith, G. Boer, P. Gonzalez, V. Kharin, et al. "A Verification Framework for Interannual-to-Decadal Predictions Experiments." Climate Dynamics 40, no. 1–2 (January 1, 2013): 245–72. https://doi.org/10/f4jjvf.

User Guide

- *Setting Up Your Dataset*
- *PredictionEnsemble Objects*
- *Comparisons*
- *Metrics*
- *Prediction Terminology*
- *Baseline Forecasts*

2.5 Setting Up Your Dataset

`climpred` relies on a consistent naming system for `xarray` dimensions. This allows things to run more easily under-the-hood.

Prediction ensembles are expected at the minimum to contain dimensions `init` and `lead`. `init` is the initialization dimension, that relays the time steps at which the ensemble was initialized. `lead` is the lead time of the forecasts from initialization. Another crucial dimension is `member`, which holds the various ensemble members. Any additional dimensions will be passed through `climpred` without issue: these could be things like `lat`, `lon`, `depth`, etc.

Control runs, references, and observational products are expected to contain the `time` dimension at the minimum. For best use of `climpred`, their `time` dimension should cover the full length of `init` from the accompanying prediction ensemble, if possible. These products can also include additional dimensions, such as `lat`, `lon`, `depth`, etc.

See the below table for a summary of dimensions used in `climpred`, and data types that `climpred` supports for them.

short_name	types	long_name
`lead`	`int`	lead timestep after initialization [`init`]
`init`	`int`	initialization: start date of experiment
`member`	`int`, `str`	ensemble member

2.6 PredictionEnsemble Objects

One of the major features of `climpred` is our objects that are based upon the `PredictionEnsemble` class. We supply users with a `HindcastEnsemble` and `PerfectModelEnsemble` object. We encourage users to take advantage of these high-level objects, which wrap all of our core functions. These objects don't comprehensively cover all functions yet, but eventually we'll deprecate direct access to the function calls in favor of the lightweight objects.

Briefly, we consider a `HindcastEnsemble` to be one that is initialized from some observational-like product (e.g., assimilated data, reanalysis products, or a model reconstruction). Thus, this object is built around comparing the initialized ensemble to various observational products. In contrast, a `PerfectModelEnsemble` is one that is initialized off of a model control simulation. These forecasting systems are not meant to be compared directly to real-world observations. Instead, they provide a contained model environment with which to theoretically study the limits of predictability. You can read more about the terminology used in `climpred` here.

Let's create a demo object to explore some of the functionality and why they are much smoother to use than direct function calls.

[1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import xarray as xr

from climpred import HindcastEnsemble
import climpred
```

We can pull in some sample data that is packaged with `climpred`.

[2]:
```
climpred.tutorial.load_dataset()
```

```
'MPI-control-1D': area averages for the MPI control run of SST/SSS.
'MPI-control-3D': lat/lon/time for the MPI control run of SST/SSS.
'MPI-PM-DP-1D': perfect model decadal prediction ensemble area averages of SST/SSS/
↪AMO.
'MPI-PM-DP-3D': perfect model decadal prediction ensemble lat/lon/time of SST/SSS/AMO.
'CESM-DP-SST': hindcast decadal prediction ensemble of global mean SSTs.
'CESM-DP-SSS': hindcast decadal prediction ensemble of global mean SSS.
'CESM-DP-SST-3D': hindcast decadal prediction ensemble of eastern Pacific SSTs.
'CESM-LE': uninitialized ensemble of global mean SSTs.
'MPIESM_miklip_baseline1-hind-SST-global': hindcast initialized ensemble of global␣
↪mean SSTs
'MPIESM_miklip_baseline1-hist-SST-global': uninitialized ensemble of global mean SSTs
'MPIESM_miklip_baseline1-assim-SST-global': assimilation in MPI-ESM of global mean␣
↪SSTs
'ERSST': observations of global mean SSTs.
'FOSI-SST': reconstruction of global mean SSTs.
'FOSI-SSS': reconstruction of global mean SSS.
'FOSI-SST-3D': reconstruction of eastern Pacific SSTs
```

2.6.1 HindcastEnsemble

We'll start out with a `HindcastEnsemble` demo, followed by a `PerfectModelEnsemble` case.

[3]:
```
hind = climpred.tutorial.load_dataset('CESM-DP-SST') # CESM-DPLE hindcast ensemble␣
↪output.
obs = climpred.tutorial.load_dataset('ERSST') # ERSST observations.
recon = climpred.tutorial.load_dataset('FOSI-SST') # Reconstruction simulation that␣
↪initialized CESM-DPLE.
```

CESM-DPLE was drift-corrected prior to uploading the output, so we just need to subtract the climatology over the same period for our other products before building the object.

[4]:
```
obs = obs - obs.sel(time=slice(1964, 2014)).mean('time')
recon = recon - recon.sel(time=slice(1964, 2014)).mean('time')
```

Now we instantiate the `HindcastEnsemble` object and append all of our products to it.

[5]:
```
hindcast = HindcastEnsemble(hind) # Instantiate object by passing in our initialized␣
↪ensemble.
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 ...
References:
    None
Uninitialized:
    None
```

Now we just use the `add_` methods to attach other objects. See the API here. **Note that we strive to make our conventions follow those of "xarray"'s**. For example, we don't allow inplace operations. One has to run `hindcast = hindcast.add_reference(...)` to modify the object upon later calls rather than just `hindcast.add_reference(...)`.

[6]:
```
hindcast = hindcast.add_reference(recon, 'reconstruction')
hindcast = hindcast.add_reference(obs, 'ERSST')
```

[7]:
```
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 ...
reconstruction:
    SST      (time) float64 -0.05064 -0.0868 -0.1396 ... 0.3023 0.3718 0.292
ERSST:
    SST      (time) float32 -0.40146065 -0.35238647 ... 0.34601402 0.45021248
Uninitialized:
    None
```

You can apply most standard `xarray` functions directly to our objects! `climpred` will loop through the objects and apply the function to all applicable `xarray.Datasets` within the object. If you reference a dimension that doesn't exist for the given `xarray.Dataset`, it will ignore it. This is useful, since the initialized ensemble is expected to have dimension `init`, while other products have dimension `time` (see more here).

Let's start by taking the ensemble mean of the initialized ensemble so our metric computations don't have to take the extra time on that later. I'm just going to use deterministic metrics here, so we don't need the individual ensemble members. Note that above our initialized ensemble had a `member` dimension, and now it is reduced.

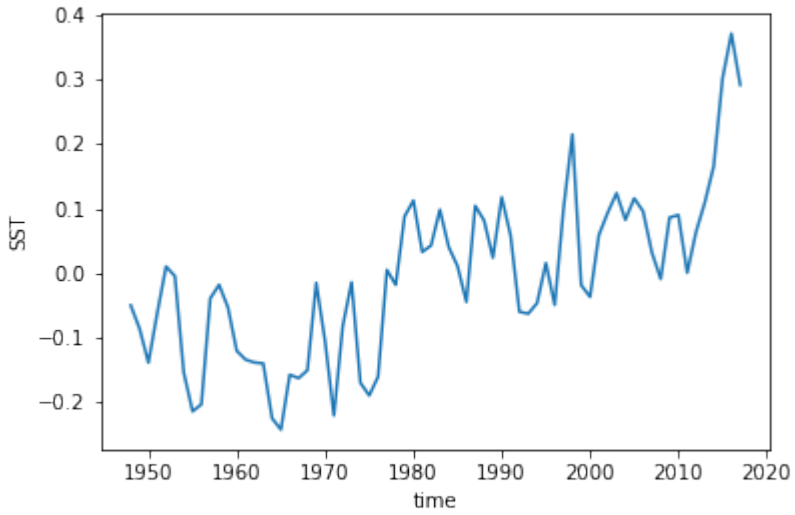
[8]:
```
hindcast = hindcast.mean('member')
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead) float64 -0.2121 -0.1637 -0.1206 ... 0.7286 0.7532
reconstruction:
    SST      (time) float64 -0.05064 -0.0868 -0.1396 ... 0.3023 0.3718 0.292
ERSST:
    SST      (time) float32 -0.40146065 -0.35238647 ... 0.34601402 0.45021248
Uninitialized:
    None
```

We still have a trend in all of our products, so we could also detrend them as well.

[9]:
```
hindcast.get_reference('reconstruction').SST.plot()
```

[9]:
```
[<matplotlib.lines.Line2D at 0x128348dd8>]
```



[10]:
```
from scipy.signal import detrend
```

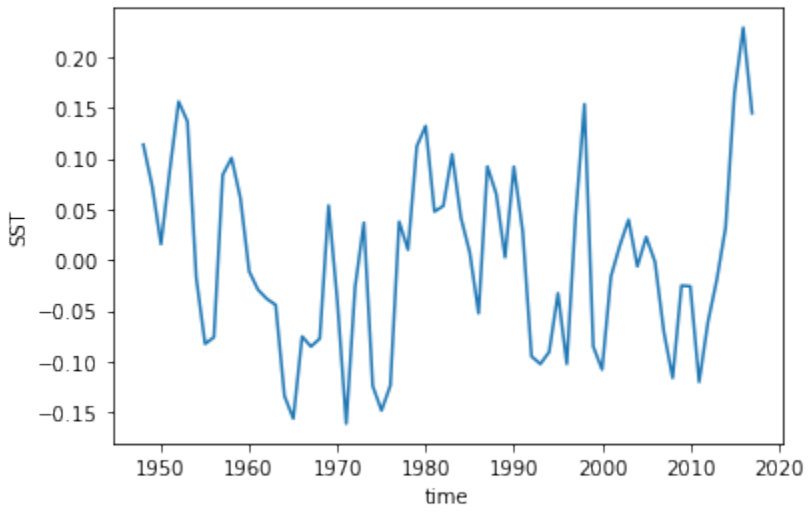
I'm going to transpose this first since my initialized ensemble has dimensions ordered `(init, lead)` and `scipy.signal.detrend` is applied over the last axis. I'd like to detrend over the `init` dimension rather than `lead` dimension.

[11]:
```
hindcast = hindcast.transpose().apply(detrend)
```

And it looks like everything got detrended by a linear fit! That wasn't too hard.

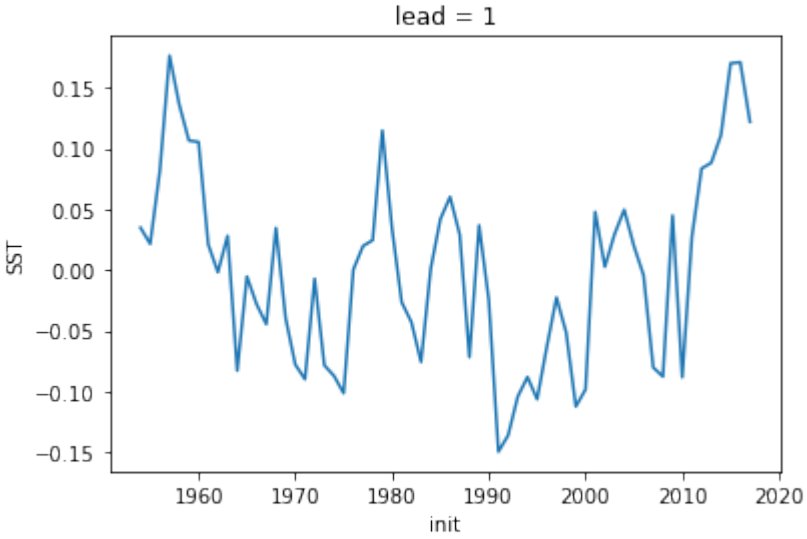
[12]:
```
hindcast.get_reference('reconstruction').SST.plot()
```

[12]:
```
[<matplotlib.lines.Line2D at 0x128422588>]
```



[13]:
```
hindcast.get_initialized().isel(lead=0).SST.plot()
```

[13]:
```
[<matplotlib.lines.Line2D at 0x10e971470>]
```



Now that we've done our pre-processing, let's quickly compute some metrics. Check the metrics page here for all the keywords you can use. The API is currently pretty simple for the `HindcastEnsemble`. You can essentially compute standard skill metrics and a reference persistence forecast.

If you just pass a metric, it'll compute the skill metric against all references and return a dictionary with keys of the names the user entered when adding them.

[14]:
```
hindcast.compute_metric(metric='mse')
```

[14]:
```
{'reconstruction': <xarray.Dataset>
 Dimensions:  (lead: 10)
 Coordinates:
   * lead     (lead) int32 1 2 3 4 5 6 7 8 9 10
 Data variables:
     SST      (lead) float64 0.005091 0.009096 0.008964 ... 0.01103 0.01261
 Attributes:
     prediction_skill:              calculated by climpred https://climpred.re...
     skill_calculated_by_function:  compute_hindcast
     number_of_initializations:     64
     metric:                        mse
     comparison:                    e2r
     created:                       2019-12-29 12:39:46,
 'ERSST': <xarray.Dataset>
 Dimensions:  (lead: 10)
 Coordinates:
   * lead     (lead) int32 1 2 3 4 5 6 7 8 9 10
 Data variables:
     SST      (lead) float64 0.003606 0.005651 0.006373 ... 0.007823 0.009009
 Attributes:
     prediction_skill:              calculated by climpred https://climpred.re...
     skill_calculated_by_function:  compute_hindcast
     number_of_initializations:     64
     metric:                        mse
     comparison:                    e2r
     created:                       2019-12-29 12:39:46}
```

One can also directly call individual references to compare to. Here we leverage `xarray`'s plotting method to compute Mean Absolute Error and the Anomaly Correlation Coefficient for both our reference products, as well as the

equivalent metrics computed for persistence forecasts for each of those metrics.

[15]:
```
import numpy as np

plt.style.use('ggplot')
plt.style.use('seaborn-talk')

RECON_COLOR = '#1b9e77'
OBS_COLOR = '#7570b3'

f, axs = plt.subplots(nrows=2, figsize=(8, 8), sharex=True)

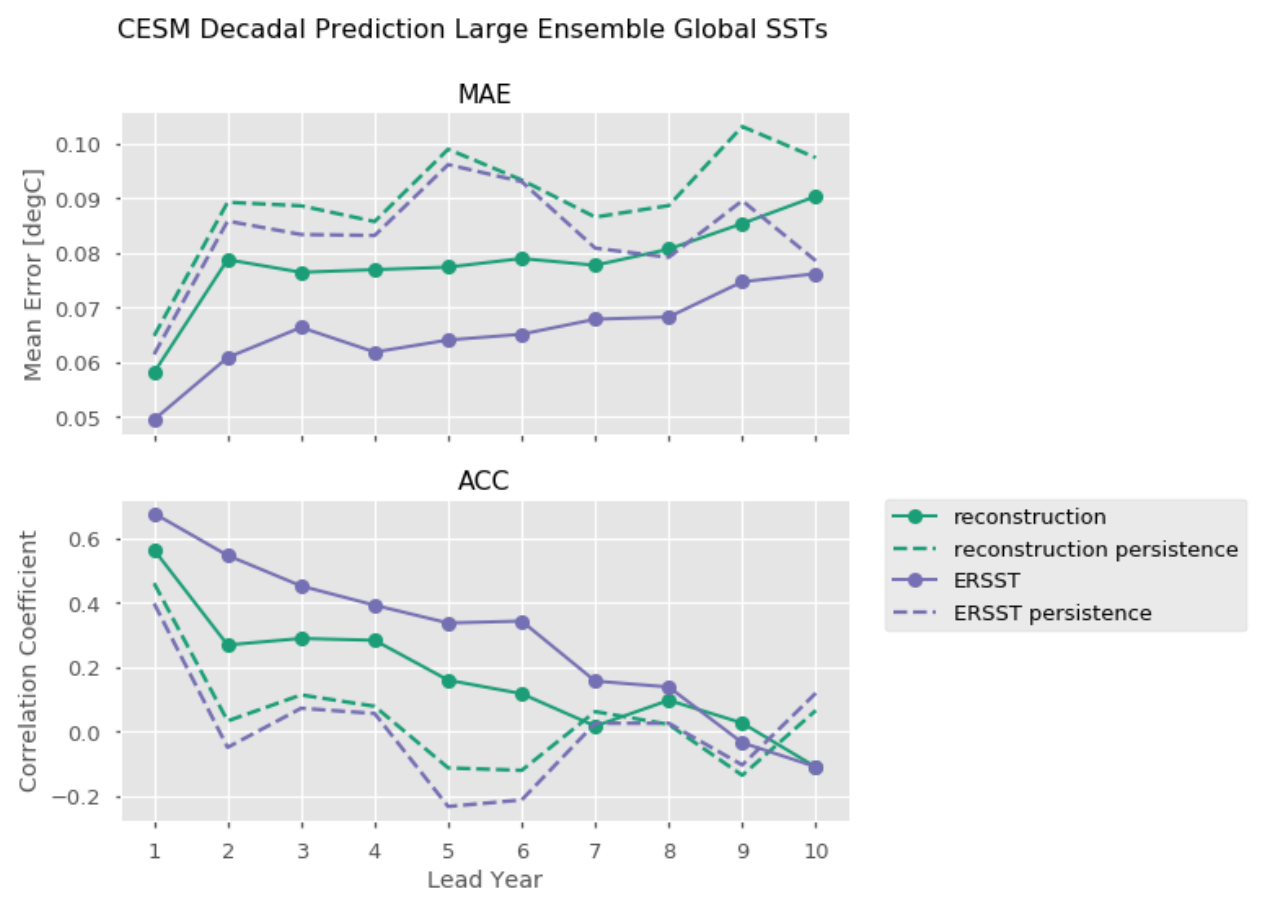
for ax, metric in zip(axs.ravel(), ['mae', 'acc']):
    handles = []
    for product, color in zip(['reconstruction', 'ERSST'], [RECON_COLOR, OBS_COLOR]):
        p1, = hindcast.compute_metric(product, metric=metric).SST.plot(ax=ax,
                                                                       marker='o',
                                                                       color=color,
                                                                       label=product,
                                                                       linewidth=2)
        p2, = hindcast.compute_persistence(product, metric=metric).SST.plot(ax=ax,
                                                                            color=color,
                                                                            linestyle='--',
                                                                            label=product +
↪' persistence')
        handles.append(p1)
        handles.append(p2)
    ax.set_title(metric.upper())


axs[0].set_ylabel('Mean Error [degC]')
axs[1].set_ylabel('Correlation Coefficient')
axs[0].set_xlabel('')
axs[1].set_xlabel('Lead Year')
axs[1].set_xticks(np.arange(10)+1)

# matplotlib/xarray returning weirdness for the legend handles.
handles = [i.get_label() for i in handles]

# a little trick to put the legend on the outside.
plt.legend(handles, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)

plt.suptitle('CESM Decadal Prediction Large Ensemble Global SSTs', fontsize=16)
plt.show()
```



2.6.2 PerfectModelEnsemble

We'll now play around a bit with the `PerfectModelEnsemble` object, using sample data from the MPI perfect model configuration.

[16]:
```
from climpred import PerfectModelEnsemble
```

[17]:
```
ds = climpred.tutorial.load_dataset('MPI-PM-DP-1D') # initialized ensemble from MPI
control = climpred.tutorial.load_dataset('MPI-control-1D') # base control run that␣
↪initialized it
```

[18]:
```
print(ds)
```

```
<xarray.Dataset>
Dimensions:  (area: 3, init: 12, lead: 20, member: 10, period: 5)
Coordinates:
  * lead     (lead) int64 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
  * period   (period) object 'DJF' 'JJA' 'MAM' 'SON' 'ym'
  * area     (area) object 'global' 'North_Atlantic' 'North_Atlantic_SPG'
  * init     (init) int64 3014 3023 3045 3061 3124 ... 3175 3178 3228 3237 3257
  * member   (member) int64 0 1 2 3 4 5 6 7 8 9
Data variables:
    tos      (period, lead, area, init, member) float32 ...
```

(continues on next page)

(continued from previous page)

```
    sos      (period, lead, area, init, member) float32 ...
    AMO      (period, lead, area, init, member) float32 ...
```

[19]:
```
pm = climpred.PerfectModelEnsemble(ds)
pm = pm.add_control(control)
print(pm)
```

```
<climpred.PerfectModelEnsemble>
Initialized Ensemble:
    tos      (period, lead, area, init, member) float32 ...
    sos      (period, lead, area, init, member) float32 ...
    AMO      (period, lead, area, init, member) float32 ...
Control:
    tos      (period, time, area) float32 ...
    sos      (period, time, area) float32 ...
    AMO      (period, time, area) float32 ...
Uninitialized:
    None
```

Our objects are carrying sea surface temperature (`tos`), sea surface salinity (`sos`), and the Atlantic Multidecadal Oscillation index (`AMO`). Say we just want to look at skill metrics for temperature and salinity over the North Atlantic in JJA. We can just call a few easy `xarray` commands to filter down our object.

[20]:
```
pm = pm.drop('AMO').sel(area='North_Atlantic', period='JJA')
```

Now we can easily compute for a host of metrics. Here I just show a number of deterministic skill metrics comparing all individual members to the initialized ensemble mean. See comparisons for more information on the `comparison` keyword.

[22]:
```
METRICS = ['mse', 'rmse', 'mae', 'acc',
           'nmse', 'nrmse', 'nmae', 'msss']

result = []
for metric in METRICS:
    result.append(pm.compute_metric(metric, comparison='m2e'))

result = xr.concat(result, 'metric')
result['metric'] = METRICS

# Leverage the `xarray` plotting wrapper to plot all results at once.
result.to_array().plot(col='metric',
                       hue='variable',
                       col_wrap=4,
                       sharey=False,
                       sharex=True)
```

[22]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1237162b0>
```



It is useful to compare the initialized ensemble to an uninitialized run. See terminology for a description on "uninitialized" simulations. This gives us information about how *initializations* lead to enhanced predictability over knowledge of external forcing, whereas a comparison to persistence just tells us how well a dynamical forecast simulation does in comparison to a naive method. We can use the `generate_uninitialized()` method to bootstrap the control run and create a pseudo-ensemble that approximates what an uninitialized ensemble would look like.

[23]:
```
pm = pm.generate_uninitialized()
print(pm)
```

```
<climpred.PerfectModelEnsemble>
Initialized Ensemble:
    tos      (lead, init, member) float32 ...
    sos      (lead, init, member) float32 ...
Control:
    tos      (time) float32 ...
    sos      (time) float32 ...
Uninitialized:
    tos      (init, member, lead) float32 12.829465 13.21901 ... 13.952087
    sos      (init, member, lead) float32 33.180943 33.192753 ... 33.15761
```

[24]:
```
pm = pm.drop('tos') # Just assess for salinity.
```

Here I plot the ACC for the initialized, uninitialized, and persistence forecasts for North Atlantic sea surface salinity in JJA. I add circles to the lines if the correlations are statistically significant for $p <= 0.05$.

[25]:
```
# ACC for initialized ensemble
acc = pm.compute_metric('acc')
acc.sos.plot(color='red')
acc.where(pm.compute_metric('p_pval') <= 0.05).sos.plot(marker='o', linestyle='None',␣
↪color='red', label='initialized')

# ACC for 'uninitialized' ensemble
acc = pm.compute_uninitialized('acc')
acc.sos.plot(color='gray')
acc.where(pm.compute_uninitialized('p_pval') <= 0.05).sos.plot(marker='o', linestyle=
↪'None', color='gray', label='uninitialized')

# ACC for persistence forecast
```

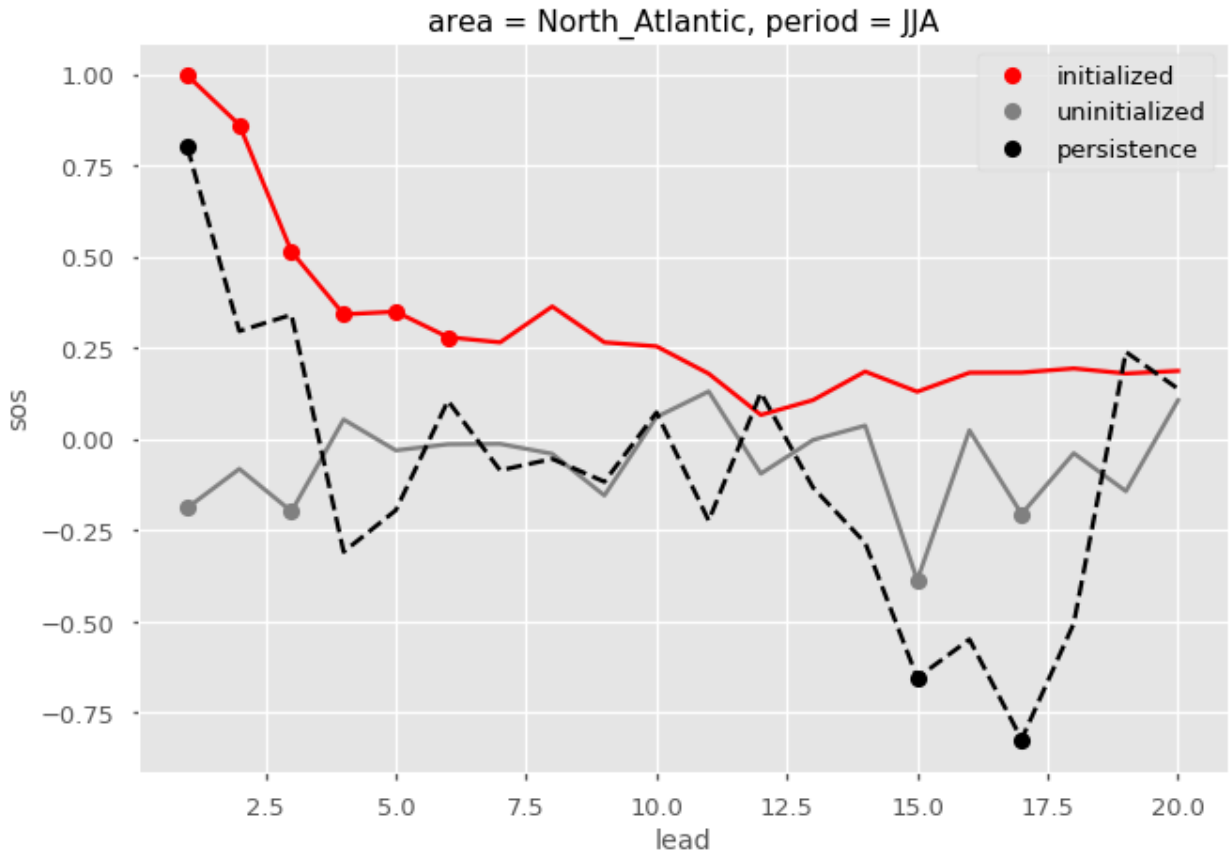
(continues on next page)

(continued from previous page)

```
acc = pm.compute_persistence('acc')
acc.sos.plot(color='k', linestyle='--')
acc.where(pm.compute_persistence('p_pval') <= 0.05).sos.plot(marker='o', linestyle=
↪'None', color='k', label='persistence')

plt.legend()
```

[25]:
```
<matplotlib.legend.Legend at 0x12d14dcf8>
```



2.7 Metrics

All high-level functions have an optional `metric` argument that can be called to determine which metric is used in computing predictability.

Note: We use the phrase 'observations' `o` here to refer to the 'truth' data to which we compare the forecast `f`. These metrics can also be applied in reference to a control simulation, reconstruction, observations, etc. This would just change the resulting score from referencing skill to referencing potential predictability.

Internally, all metric functions require `forecast` and `reference` as inputs. The dimension `dim` is set internally by *`compute_hindcast()`* or *`compute_perfect_model()`* to specify over which dimensions the `metric` is applied. See *Comparisons* for more on the `dim` argument.

2.7.1 Deterministic

Deterministic metrics assess the forecast as a definite prediction of the future, rather than in terms of probabilities. Another way to look at deterministic metrics is that they are a special case of probabilistic metrics where a value of one is assigned to one category and zero to all others [Jolliffe2011].

Correlation Metrics

The below metrics rely fundamentally on correlations in their computation. In the literature, correlation metrics are typically referred to as the Anomaly Correlation Coefficient (ACC). This implies that anomalies in the forecast and observations are being correlated. Typically, this is computed using the linear *Pearson Product-Moment Correlation*. However, `climpred` also offers the *Spearman's Rank Correlation*.

Note that the p value associated with these correlations is computed via a separate metric. Use `pearson_r_p_value` or `spearman_r_p_value` to compute p values assuming that all samples in the correlated time series are independent. Use `pearson_r_eff_p_value` or `spearman_r_eff_p_value` to account for autocorrelation in the time series by calculating the `effective_sample_size`.

Pearson Product-Moment Correlation Coefficient

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [1]: print(f"\n\nKeywords: {metric_aliases['pearson_r']}")


Keywords: ['pearson_r', 'pr', 'acc', 'pacc']
```

`climpred.metrics.`**`_pearson_r`**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Pearson product-moment correlation coefficient.

A measure of the linear association between the forecast and observations that is independent of the mean and variance of the individual distributions. This is also known as the Anomaly Correlation Coefficient (ACC) when correlating anomalies.

$$corr = \frac{cov(f, o)}{\sigma_f \cdot \sigma_o},$$

where σ_f and σ_o represent the standard deviation of the forecast and observations over the experimental period, respectively.

Note: Use metric `pearson_r_p_value` or `pearson_r_eff_p_value` to get the corresponding p value.

Parameters

- **`forecast`** (*`xarray object`*) – Forecast.
- **`reference`** (*`xarray object`*) – Reference (e.g. observations, control run).
- **`dim`** (*`str`*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **`weights`** (*`xarray object, optional`*) – Weights to apply over dimension. Defaults to `None`.
- **`skipna`** (*`bool, optional`*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	-1.0
maximum	1.0
perfect	1.0
orientation	positive

See also:

- xskillscore.pearson_r
- xskillscore.pearson_r_p_value
- climpred.pearson_r_p_value
- climpred.pearson_r_eff_p_value

Pearson Correlation p value

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [2]: print(f"\n\nKeywords: {metric_aliases['pearson_r_p_value']}")


Keywords: ['pearson_r_p_value', 'p_pval', 'pvalue', 'pval']
```

climpred.metrics.**_pearson_r_p_value**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Probability that forecast and reference are linearly uncorrelated.

Two-tailed p value associated with the Pearson product-moment correlation coefficient (`pearson_r`), assuming that all samples are independent. Use `pearson_r_eff_p_value` to account for autocorrelation in the forecast and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	1.0
orientation	negative

See also:

- xskillscore.pearson_r

- xskillscore.pearson_r_p_value
- climpred.pearson_r
- climpred.pearson_r_eff_p_value

Effective Sample Size

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [3]: print(f"\n\nKeywords: {metric_aliases['effective_sample_size']}")


Keywords: ['effective_sample_size', 'n_eff', 'eff_n']
```

climpred.metrics.**_effective_sample_size**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Effective sample size for temporally correlated data.

Note: Weights are not included here due to the dependence on temporal autocorrelation.

The effective sample size extracts the number of independent samples between two time series being correlated. This is derived by assessing the magnitude of the lag-1 autocorrelation coefficient in each of the time series being correlated. A higher autocorrelation induces a lower effective sample size which raises the correlation coefficient for a given p value.

The effective sample size is used in computing the effective p value. See `pearson_r_eff_p_value` and `spearman_r_eff_p_value`.

$$N_{eff} = N \left(\frac{1 - \rho_f \rho_o}{1 + \rho_f \rho_o} \right),$$

where ρ_f and ρ_o are the lag-1 autocorrelation coefficients for the forecast and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	∞
perfect	N/A
orientation	positive

References

- Bretherton, Christopher S., et al. "The effective number of spatial degrees of freedom of a time-varying field." Journal of climate 12.7 (1999): 1990-2009.

Pearson Correlation Effective p value

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [4]: print(f"\n\nKeywords: {metric_aliases['pearson_r_eff_p_value']}")


Keywords: ['pearson_r_eff_p_value', 'p_pval_eff', 'pvalue_eff', 'pval_eff']
```

climpred.metrics.**_pearson_r_eff_p_value**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Probability that forecast and reference are linearly uncorrelated, accounting for autocorrelation.

Note: Weights are not included here due to the dependence on temporal autocorrelation.

The effective p value is computed by replacing the sample size N in the t-statistic with the effective sample size, N_{eff}. The same Pearson product-moment correlation coefficient r is used as when computing the standard p value.

$$t = r\sqrt{\frac{N_{eff} - 2}{1 - r^2}},$$

where N_{eff} is computed via the autocorrelation in the forecast and observations.

$$N_{eff} = N\left(\frac{1 - \rho_f \rho_o}{1 + \rho_f \rho_o}\right),$$

where ρ_f and ρ_o are the lag-1 autocorrelation coefficients for the forecast and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	1.0
orientation	negative

See also:

- climpred.effective_sample_size
- climpred.spearman_r_eff_p_value

References

- Bretherton, Christopher S., et al. "The effective number of spatial degrees of freedom of a time-varying field." Journal of climate 12.7 (1999): 1990-2009.

Spearman's Rank Correlation Coefficient

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [5]: print(f"\n\nKeywords: {metric_aliases['spearman_r']}")


Keywords: ['spearman_r', 'sacc', 'sr']
```

climpred.metrics.**_spearman_r**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Spearman's rank correlation coefficient.

$$corr = \text{pearsonr}(ranked(f), ranked(o))$$

This correlation coefficient is nonparametric and assesses how well the relationship between the forecast and observations can be described using a monotonic function. It is computed by first ranking the forecasts and observations, and then correlating those ranks using the `pearson_r` correlation.

This is also known as the anomaly correlation coefficient (ACC) when comparing anomalies, although the Pearson product-moment correlation coefficient (`pearson_r`) is typically used when computing the ACC.

Note: Use metric `spearman_r_p_value` or `spearman_r_eff_p_value` to get the corresponding p value.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	-1.0
maximum	1.0
perfect	1.0
orientation	positive

See also:

- xskillscore.spearman_r
- xskillscore.spearman_r_p_value
- climpred.spearman_r_p_value
- climpred.spearman_r_eff_p_value

Spearman's Rank Correlation Coefficient p value

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [6]: print(f"\n\nKeywords: {metric_aliases['spearman_r_p_value']}")


Keywords: ['spearman_r_p_value', 's_pval', 'spvalue', 'spval']
```

climpred.metrics.**_spearman_r_p_value**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Probability that forecast and reference are monotonically uncorrelated.

Two-tailed p value associated with the Spearman's rank correlation coefficient (`spearman_r`), assuming that all samples are independent. Use `spearman_r_eff_p_value` to account for autocorrelation in the forecast and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	1.0
orientation	negative

See also:

- xskillscore.spearman_r
- xskillscore.spearman_r_p_value
- climpred.spearman_r
- climpred.spearman_r_eff_p_value

Spearman's Rank Correlation Effective p value

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [7]: print(f"\n\nKeywords: {metric_aliases['spearman_r_eff_p_value']}")


Keywords: ['spearman_r_eff_p_value', 's_pval_eff', 'spvalue_eff', 'spval_eff']
```

climpred.metrics.**_spearman_r_eff_p_value**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Probability that forecast and reference are monotonically uncorrelated, accounting for autocorrelation.

Note: Weights are not included here due to the dependence on temporal autocorrelation.

The effective p value is computed by replacing the sample size N in the t-statistic with the effective sample size, N_{eff}. The same Spearman's rank correlation coefficient r is used as when computing the standard p value.

$$t = r\sqrt{\frac{N_{eff} - 2}{1 - r^2}},$$

where N_{eff} is computed via the autocorrelation in the forecast and observations.

$$N_{eff} = N\left(\frac{1 - \rho_f \rho_o}{1 + \rho_f \rho_o}\right),$$

where ρ_f and ρ_o are the lag-1 autocorrelation coefficients for the forecast and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	1.0
orientation	negative

See also:

- climpred.effective_sample_size
- climpred.pearson_r_eff_p_value

References

- Bretherton, Christopher S., et al. "The effective number of spatial degrees of freedom of a time-varying field." Journal of climate 12.7 (1999): 1990-2009.

Distance Metrics

This class of metrics simply measures the distance (or difference) between forecasted values and observed values.

Mean Squared Error (MSE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [8]: print(f"\n\nKeywords: {metric_aliases['mse']}")


Keywords: ['mse']
```

climpred.metrics.**_mse**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Mean Sqaure Error (MSE).

$$MSE = \overline{(f - o)^2}$$

The average of the squared difference between forecasts and observations. This incorporates both the variance and bias of the estimator. Because the error is squared, it is more sensitive to large forecast errors than `mae`, and thus a more conservative metric. For example, a single error of 2°C counts the same as two 1°C errors when using `mae`. On the other hand, the 2°C error counts double for `mse`. See Jolliffe and Stephenson, 2011.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative

See also:

- xskillscore.mse

References

- Ian T. Jolliffe and David B. Stephenson. Forecast Verification: A Practitioner's Guide in Atmospheric Science. John Wiley & Sons, Ltd, Chichester, UK, December 2011. ISBN 978-1-119-96000-3 978-0-470-66071-3. URL: http://doi.wiley.com/10.1002/9781119960003.

Root Mean Square Error (RMSE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [9]: print(f"\n\nKeywords: {metric_aliases['rmse']}")


Keywords: ['rmse']
```

climpred.metrics.**_rmse**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Root Mean Sqaure Error (RMSE).

$$RMSE = \sqrt{\overline{(f-o)^2}}$$

The square root of the average of the squared differences between forecasts and observations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to None.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to False.

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative

See also:

- xskillscore.rmse

Mean Absolute Error (MAE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [10]: print(f"\n\nKeywords: {metric_aliases['mae']}")


Keywords: ['mae']
```

climpred.metrics.**_mae**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Mean Absolute Error (MAE).

$$MAE = \overline{|f-o|}$$

The average of the absolute differences between forecasts and observations. A more robust measure of forecast accuracy than mse which is sensitive to large outlier forecast errors.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to None.

- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative

See also:

- xskillscore.mae

References

- Ian T. Jolliffe and David B. Stephenson. Forecast Verification: A Practitioner's Guide in Atmospheric Science. John Wiley & Sons, Ltd, Chichester, UK, December 2011. ISBN 978-1-119-96000-3 978-0-470-66071-3. URL: http://doi.wiley.com/10.1002/9781119960003.

Median Absolute Error

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [11]: print(f"\n\nKeywords: {metric_aliases['median_absolute_error']}")


Keywords: ['median_absolute_error']
```

climpred.metrics.**_median_absolute_error**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Median Absolute Error.

$$median(|f - o|)$$

The median of the absolute differences between forecasts and observations. Applying the median function to absolute error makes it more robust to outliers.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative

See also:

- xskillscore.median_absolute_error

Normalized Distance Metrics

Distance metrics like `mse` can be normalized to 1. The normalization factor depends on the comparison type choosen. For example, the distance between an ensemble member and the ensemble mean is half the distance of an ensemble member with other ensemble members. See *_get_norm_factor()*.

Normalized Mean Square Error (NMSE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [12]: print(f"\n\nKeywords: {metric_aliases['nmse']}")


Keywords: ['nmse', 'nev']
```

`climpred.metrics.`**`_nmse`**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Normalized MSE (NMSE), also known as Normalized Ensemble Variance (NEV).

Mean Square Error (`mse`) normalized by the variance of the observations.

$$NMSE = NEV = \frac{MSE}{\sigma_o^2 \cdot fac} = \frac{\overline{(f-o)^2}}{\sigma_o^2 \cdot fac},$$

where fac is 1 when using comparisons involving the ensemble mean (`m2e`, `e2c`, `e2r`) and 2 when using comparisons involving individual ensemble members (`m2c`, `m2m`, `m2r`). See *_get_norm_factor()*.

Note: `climpred` uses a single-valued internal reference forecast for the NMSE, in the terminology of Murphy 1988. I.e., we use a single climatological variance of the reference *within* the experimental window for normalizing MSE.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.
- **comparison** (*str*) – Name comparison needed for normalization factor *fac*, see *_get_norm_factor()* (Handled internally by the compute functions)

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative
better than climatology	0.0 - 1.0
worse than climatology	> 1.0

References

- Griffies, S. M., and K. Bryan. "A Predictability Study of Simulated North Atlantic Multidecadal Variability." Climate Dynamics 13, no. 7–8 (August 1, 1997): 459–87. https://doi.org/10/ch4kc4.
- Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.

Normalized Mean Absolute Error (NMAE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [13]: print(f"\n\nKeywords: {metric_aliases['nmae']}")


Keywords: ['nmae']
```

climpred.metrics.**_nmae**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Normalized Mean Absolute Error (NMAE).

Mean Absolute Error (`mae`) normalized by the standard deviation of the observations.

$$NMAE = \frac{MAE}{\sigma_o \cdot fac} = \frac{\overline{|f - o|}}{\sigma_o \cdot fac},$$

where fac is 1 when using comparisons involving the ensemble mean (`m2e`, `e2c`, `e2r`) and 2 when using comparisons involving individual ensemble members (`m2c`, `m2m`, `m2r`). See *`_get_norm_factor()`*.

Note: `climpred` uses a single-valued internal reference forecast for the NMAE, in the terminology of Murphy 1988. I.e., we use a single climatological standard deviation of the reference *within* the experimental window for normalizing MAE.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

- **comparison** (*str*) – Name comparison needed for normalization factor *fac*, see *_get_norm_factor()* (Handled internally by the compute functions)

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative
better than climatology	0.0 - 1.0
worse than climatology	> 1.0

References

- Griffies, S. M., and K. Bryan. "A Predictability Study of Simulated North Atlantic Multidecadal Variability." Climate Dynamics 13, no. 7–8 (August 1, 1997): 459–87. https://doi.org/10/ch4kc4.
- Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.

Normalized Root Mean Square Error (NRMSE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [14]: print(f"\n\nKeywords: {metric_aliases['nrmse']}")


Keywords: ['nrmse']
```

climpred.metrics.**_nrmse**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Normalized Root Mean Square Error (NRMSE).

Root Mean Square Error (`rmse`) normalized by the standard deviation of the observations.

$$NRMSE = \frac{RMSE}{\sigma_o \cdot \sqrt{fac}} = \sqrt{\frac{MSE}{\sigma_o^2 \cdot fac}} = \sqrt{\frac{\overline{(f-o)^2}}{\sigma_o^2 \cdot fac}},$$

where fac is 1 when using comparisons involving the ensemble mean (`m2e`, `e2c`, `e2r`) and 2 when using comparisons involving individual ensemble members (`m2c`, `m2m`, `m2r`). See *_get_norm_factor()*.

Note: `climpred` uses a single-valued internal reference forecast for the NRMSE, in the terminology of Murphy 1988. I.e., we use a single climatological variance of the reference *within* the experimental window for normalizing RMSE.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.

- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to None.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to False.
- **comparison** (*str*) – Name comparison needed for normalization factor *fac*, see *_get_norm_factor()* (Handled internally by the compute functions)

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative
better than climatology	0.0 - 1.0
worse than climatology	> 1.0

References

- Bushuk, Mitchell, Rym Msadek, Michael Winton, Gabriel Vecchi, Xiaosong Yang, Anthony Rosati, and Rich Gudgel. "Regional Arctic Sea–Ice Prediction: Potential versus Operational Seasonal Forecast Skill." Climate Dynamics, June 9, 2018. https://doi.org/10/gd7hfq.
- Hawkins, Ed, Steffen Tietsche, Jonathan J. Day, Nathanael Melia, Keith Haines, and Sarah Keeley. "Aspects of Designing and Evaluating Seasonal-to-Interannual Arctic Sea-Ice Prediction Systems." Quarterly Journal of the Royal Meteorological Society 142, no. 695 (January 1, 2016): 672–83. https://doi.org/10/gfb3pn.
- Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.

Mean Square Error Skill Score (MSESS)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [15]: print(f"\n\nKeywords: {metric_aliases['msess']}")


Keywords: ['msess', 'ppp', 'msss']
```

climpred.metrics.**_msess**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Mean Squared Error Skill Score (MSESS).

$$MSESS = 1 - \frac{MSE}{\sigma^2_{ref} \cdot fac} = 1 - \frac{\overline{(f-o)^2}}{\sigma^2_{ref} \cdot fac},$$

where fac is 1 when using comparisons involving the ensemble mean (m2e, e2c, e2r) and 2 when using comparisons involving individual ensemble members (m2c, m2m, m2r). See *_get_norm_factor()*.

This skill score can be intepreted as a percentage improvement in accuracy. I.e., it can be multiplied by 100%.

Note: climpred uses a single-valued internal reference forecast for the MSSS, in the terminology of Murphy 1988. I.e., we use a single climatological variance of the reference *within* the experimental window for

normalizing MSE.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.
- **comparison** (*str*) – Name comparison needed for normalization factor *fac*, see *_get_norm_factor()* (Handled internally by the compute functions)

Details:

minimum	-∞
maximum	1.0
perfect	1.0
orientation	positive
better than climatology	> 0.0
equal to climatology	0.0
worse than climatology	< 0.0

References

- Griffies, S. M., and K. Bryan. "A Predictability Study of Simulated North Atlantic Multidecadal Variability." Climate Dynamics 13, no. 7–8 (August 1, 1997): 459–87. https://doi.org/10/ch4kc4.
- Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.
- Pohlmann, Holger, Michael Botzet, Mojib Latif, Andreas Roesch, Martin Wild, and Peter Tschuck. "Estimating the Decadal Predictability of a Coupled AOGCM." Journal of Climate 17, no. 22 (November 1, 2004): 4463–72. https://doi.org/10/d2qf62.
- Bushuk, Mitchell, Rym Msadek, Michael Winton, Gabriel Vecchi, Xiaosong Yang, Anthony Rosati, and Rich Gudgel. "Regional Arctic Sea–Ice Prediction: Potential versus Operational Seasonal Forecast Skill. Climate Dynamics, June 9, 2018. https://doi.org/10/gd7hfq.

Mean Absolute Percentage Error (MAPE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [16]: print(f"\n\nKeywords: {metric_aliases['mape']}")


Keywords: ['mape']
```

climpred.metrics.**_mape**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Mean Absolute Percentage Error (MAPE).

Mean absolute error (`mae`) expressed as a percentage error relative to the observations.

$$MAPE = \frac{1}{n}\sum\frac{|f-o|}{|o|}$$

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	0.0
orientation	negative

See also:

- xskillscore.mape

Symmetric Mean Absolute Percentage Error (sMAPE)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [17]: print(f"\n\nKeywords: {metric_aliases['smape']}")


Keywords: ['smape']
```

climpred.metrics.**_smape**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Symmetric Mean Absolute Percentage Error (sMAPE).

Similar to the Mean Absolute Percentage Error (`mape`), but sums the forecast and observation mean in the denominator.

$$sMAPE = \frac{1}{n}\sum\frac{|f-o|}{|f|+|o|}$$

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.

- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	0.0
maximum	1.0
perfect	0.0
orientation	negative

See also:

- xskillscore.smape

Unbiased Anomaly Correlation Coefficient (uACC)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [18]: print(f"\n\nKeywords: {metric_aliases['uacc']}")


Keywords: ['uacc']
```

climpred.metrics.**_uacc**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Bushuk's unbiased Anomaly Correlation Coefficient (uACC).

This is typically used in perfect model studies. Because the perfect model Anomaly Correlation Coefficient (ACC) is strongly state dependent, a standard ACC (e.g. one computed using `pearson_r`) will be highly sensitive to the set of start dates chosen for the perfect model study. The Mean Square Skill Score (`MSSS`) can be related directly to the ACC as `MSSS = ACC^(2)` (see Murphy 1988 and Bushuk et al. 2019), so the unbiased ACC can be derived as `uACC = sqrt(MSSS)`.

$$uACC = \sqrt{MSSS} = \sqrt{1 - \frac{\overline{(f-o)^2}}{\sigma^2_{ref} \cdot fac}},$$

where fac is 1 when using comparisons involving the ensemble mean (`m2e`, `e2c`, `e2r`) and 2 when using comparisons involving individual ensemble members (`m2c`, `m2m`, `m2r`). See *_get_norm_factor()*.

Note: Because of the square root involved, any negative `MSSS` values are automatically converted to NaNs.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.
- **comparison** (*str*) – Name comparison needed for normalization factor `fac`, see *_get_norm_factor()* (Handled internally by the compute functions)

Details:

minimum	0.0
maximum	1.0
perfect	1.0
orientation	positive
better than climatology	> 0.0
equal to climatology	0.0

References

- Bushuk, Mitchell, Rym Msadek, Michael Winton, Gabriel Vecchi, Xiaosong Yang, Anthony Rosati, and Rich Gudgel. "Regional Arctic Sea–Ice Prediction: Potential versus Operational Seasonal Forecast Skill." Climate Dynamics, June 9, 2018. https://doi.org/10/gd7hfq.
- Allan H. Murphy. Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient. Monthly Weather Review, 116(12):2417–2424, December 1988. https://doi.org/10/fc7mxd.

Murphy Decomposition Metrics

Metrics derived in [Murphy1988] which decompose the `MSESS` into a correlation term, a conditional bias term, and an unconditional bias term. See https://www-miklip.dkrz.de/about/murcss/ for a walk through of the decomposition.

Standard Ratio

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [19]: print(f"\n\nKeywords: {metric_aliases['std_ratio']}")


Keywords: ['std_ratio']
```

climpred.metrics.**_std_ratio**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Ratio of standard deviations of the forecast over the reference.

$$\text{std ratio} = \frac{\sigma_f}{\sigma_o},$$

where σ_f and σ_o are the standard deviations of the forecast and the observations over the experimental period, respectively.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute functions.

Details:

minimum	0.0
maximum	∞
perfect	1.0
orientation	N/A

References

- https://www-miklip.dkrz.de/about/murcss/

Conditional Bias

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [20]: print(f"\n\nKeywords: {metric_aliases['conditional_bias']}")


Keywords: ['conditional_bias', 'c_b', 'cond_bias']
```

climpred.metrics.**_conditional_bias**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Conditional bias between forecast and reference.

$$\text{conditional bias} = r_{fo} - \frac{\sigma_f}{\sigma_o},$$

where σ_f and σ_o are the standard deviations of the forecast and observations over the experimental period, respectively.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute functions.

Details:

minimum	-∞
maximum	1.0
perfect	0.0
orientation	negative

References

- https://www-miklip.dkrz.de/about/murcss/

Unconditional Bias

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [21]: print(f"\n\nKeywords: {metric_aliases['unconditional_bias']}")


Keywords: ['unconditional_bias', 'u_b', 'bias']
```

Simple bias of the forecast minus the observations.

climpred.metrics.**_unconditional_bias**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Unconditional bias.

$$bias = f - o$$

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute functions.

Details:

minimum	-∞
maximum	∞
perfect	0.0
orientation	negative

References

- https://www.cawcr.gov.au/projects/verification/
- https://www-miklip.dkrz.de/about/murcss/

Bias Slope

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [22]: print(f"\n\nKeywords: {metric_aliases['bias_slope']}")


Keywords: ['bias_slope']
```

climpred.metrics.**_bias_slope**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Bias slope between reference and forecast standard deviations.

$$\text{bias slope} = \frac{s_o}{s_f} \cdot r_{fo},$$

where r_{fo} is the Pearson product-moment correlation between the forecast and the observations and s_o and s_f are the standard deviations of the observations and forecast over the experimental period, respectively.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute functions.

Details:

minimum	0.0
maximum	∞
perfect	1.0
orientation	negative

References

- https://www-miklip.dkrz.de/about/murcss/

Murphy's Mean Square Error Skill Score

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [23]: print(f"\n\nKeywords: {metric_aliases['msess_murphy']}")


Keywords: ['msess_murphy', 'msss_murphy']
```

climpred.metrics.**_msess_murphy**(*forecast*, *reference*, *dim=None*, ***metric_kwargs*)

Murphy's Mean Square Error Skill Score (MSESS).

$$MSESS_{Murphy} = r_{fo}^2 - [\text{conditional bias}]^2 - [\frac{\text{(unconditional) bias}}{\sigma_o}]^2,$$

where r_{fo}^2 represents the Pearson product-moment correlation coefficient between the forecast and observations and σ_o represents the standard deviation of the observations over the experimental period. See `conditional_bias` and `unconditional_bias` for their respective formulations.

Parameters

- **forecast** (*xarray object*) – Forecast.
- **reference** (*xarray object*) – Reference (e.g. observations, control run).
- **dim** (*str*) – Dimension(s) to perform metric over. Automatically set by compute function.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	-∞
maximum	1.0
perfect	1.0
orientation	positive

See also:

- climpred.pearson_r
- climpred.conditional_bias
- climpred.unconditional_bias

References

- https://www-miklip.dkrz.de/about/murcss/
- Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.

2.7.2 Probabilistic

Probabilistic metrics include the spread of the ensemble simulations in their calculations and assign a probability value between 0 and 1 to their forecasts [Jolliffe2011].

Continuous Ranked Probability Score (CRPS)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [24]: print(f"\n\nKeywords: {metric_aliases['crps']}")


Keywords: ['crps']
```

climpred.metrics.**_crps**(*forecast*, *reference*, ***metric_kwargs*)

Continuous Ranked Probability Score (CRPS).

The CRPS can also be considered as the probabilistic Mean Absolute Error (`mae`). It compares the empirical distribution of an ensemble forecast to a scalar observation. Smaller scores indicate better skill.

$$CRPS = \int_{-\infty}^{\infty} (F(f) - H(f-o))^2 df,$$

where $F(f)$ is the cumulative distribution function (CDF) of the forecast (since the observations are not assigned a probability), and H() is the Heaviside step function where the value is 1 if the argument is positive (i.e., the forecast overestimates observations) or zero (i.e., the forecast equals observations) and is 0 otherwise (i.e., the forecast is less than observations).

Note: The CRPS is expressed in the same unit as the observed variable. It generalizes the Mean Absolute Error (MAE), and reduces to the MAE if the forecast is determinstic.

Parameters

- **forecast** (*xr.object*) – Forecast with *member* dim.
- **reference** (*xr.object*) – References without *member* dim (e.g. observations, control run).
- **metric_kwargs** (*xr.object*) – If provided, the CRPS is calculated exactly with the assigned probability weights to each forecast. Weights should be positive, but do not need to be normalized. By default, each forecast is weighted equally.

Details:

minimum	0.0
maximum	∞
perfect	0.0
orientation	negative

References

- Matheson, James E., and Robert L. Winkler. "Scoring Rules for Continuous Probability Distributions." Management Science 22, no. 10 (June 1, 1976): 1087–96. https://doi.org/10/cwwt4g.
- https://www.lokad.com/continuous-ranked-probability-score

See also:

- properscoring.crps_ensemble
- xskillscore.crps_ensemble

Continuous Ranked Probability Skill Score (CRPSS)

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [25]: print(f"\n\nKeywords: {metric_aliases['crpss']}")


Keywords: ['crpss']
```

climpred.metrics.**_crpss**(*forecast*, *reference*, ***metric_kwargs*)

Continuous Ranked Probability Skill Score.

This can be used to assess whether the ensemble spread is a useful measure for the forecast uncertainty by comparing the CRPS of the ensemble forecast to that of a reference forecast with the desired spread.

$$CRPSS = 1 - \frac{CRPS_{initialized}}{CRPS_{clim}}$$

Note: When assuming a Gaussian distribution of forecasts, use default `gaussian=True`. If not gaussian, you may specify the distribution type, xmin/xmax/tolerance for integration (see xskillscore.crps_quadrature).

Parameters

- **forecast** (*xr.object*) – Forecast with `member` dim.
- **reference** (*xr.object*) – References without `member` dim.
- **gaussian** (*bool, optional*) – If `True`, assum Gaussian distribution for baseline skill. Defaults to `True`.
- **cdf_or_dist** (*scipy.stats*) – Function which returns the cumulative density of the forecast at value x. This can also be an object with a callable `cdf()` method such as a `scipy.stats.distribution` object. Defaults to `scipy.stats.norm`.
- **xmin** (*float*) – Lower bounds for integration. Only use if not assuming Gaussian.
- **xmax** (*float*) –
- **tol** (*float, optional*) – The desired accuracy of the CRPS. Larger values will speed up integration. If `tol` is set to `None`, bounds errors or integration tolerance errors will be ignored. Only use if not assuming Gaussian.

Details:

minimum	-∞
maximum	1.0
perfect	1.0
orientation	positive
better than climatology	> 0.0
worse than climatology	< 0.0

References

- Matheson, James E., and Robert L. Winkler. "Scoring Rules for Continuous Probability Distributions." Management Science 22, no. 10 (June 1, 1976): 1087–96. https://doi.org/10/cwwt4g.
- Gneiting, Tilmann, and Adrian E Raftery. "Strictly Proper Scoring Rules, Prediction, and Estimation." Journal of the American Statistical Association 102, no. 477 (March 1, 2007): 359–78. https://doi.org/10/c6758w.

Example

```
>>> compute_perfect_model(ds, control, metric='crpss')
>>> compute_perfect_model(ds, control, metric='crpss', gaussian=False,
                          cdf_or_dist=scipy.stats.norm, xminimum=-10,
                          xmaximum=10, tol=1e-6)
```

See also:

- properscoring.crps_ensemble
- xskillscore.crps_ensemble

Continuous Ranked Probability Skill Score Ensemble Spread

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [26]: print(f"\n\nKeywords: {metric_aliases['crpss_es']}")


Keywords: ['crpss_es']
```

climpred.metrics.**_crpss_es**(*forecast*, *reference*, ***metric_kwargs*)

Continuous Ranked Probability Skill Score Ensemble Spread.

If the ensemble variance is smaller than the observed `mse`, the ensemble is said to be under-dispersive (or overconfident). An ensemble with variance larger than the observations indicates one that is over-dispersive (underconfident).

$$CRPSS = 1 - \frac{CRPS(\sigma_f^2)}{CRPS(\sigma_o^2)}$$

Parameters

- **forecast** (*xr.object*) – Forecast with `member` dim.
- **reference** (*xr.object*) – References without `member` dim.
- **weights** (*xarray object, optional*) – Weights to apply over dimension. Defaults to `None`.
- **skipna** (*bool, optional*) – If True, skip NaNs over dimension being applied to. Defaults to `False`.

Details:

minimum	-∞
maximum	0.0
perfect	0.0
orientation	positive
under-dispersive	> 0.0
over-dispersive	< 0.0

References

- Kadow, Christopher, Sebastian Illing, Oliver Kunst, Henning W. Rust, Holger Pohlmann, Wolfgang A. Müller, and Ulrich Cubasch. "Evaluation of Forecasts by Accuracy and Spread in the MiKlip Decadal Climate Prediction System." Meteorologische Zeitschrift, December 21, 2016, 631–43. https://doi.org/10/f9jrhw.

Range:

- perfect: 0
- else: negative

Brier Score

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [27]: print(f"\n\nKeywords: {metric_aliases['brier_score']}")


Keywords: ['brier_score', 'brier', 'bs']
```

climpred.metrics.**_brier_score**(*forecast*, *reference*, ***metric_kwargs*)

Brier Score.

The Mean Square Error (`mse`) of probabilistic two-category forecasts where the observations are either 0 (no occurrence) or 1 (occurrence) and forecast probability may be arbitrarily distributed between occurrence and non-occurrence. The Brier Score equals zero for perfect (single-valued) forecasts and one for forecasts that are always incorrect.

$$BS(f, o) = (f_1 - o)^2,$$

where f_1 is the forecast probability of $o = 1$.

Note: The Brier Score requires that the reference is binary, i.e., can be described as one (a "hit") or zero (a "miss").

Parameters

- **forecast** (*xr.object*) – Forecast with `member` dim.
- **reference** (*xr.object*) – References without `member` dim.
- **func** (*function*) – Function to be applied to reference and forecasts and then `mean('member')` to get forecasts and reference in interval [0,1].

Details:

minimum	0.0
maximum	1.0
perfect	0.0
orientation	negative

Reference:

- Brier, Glenn W. Verification of forecasts expressed in terms of probability." Monthly Weather Review 78, no. 1 (1950). https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.
- https://www.nws.noaa.gov/oh/rfcdev/docs/ Glossary_Forecast_Verification_Metrics.pdf

See also:

- properscoring.brier_score
- xskillscore.brier_score

Example

```
>>> def pos(x): return x > 0
>>> compute_perfect_model(ds, control, metric='brier_score', func=pos)
```

Threshold Brier Score

```
# Enter any of the below keywords in ``metric=...`` for the compute functions.
In [28]: print(f"\n\nKeywords: {metric_aliases['threshold_brier_score']}")


Keywords: ['threshold_brier_score', 'tbs']
```

climpred.metrics.**_threshold_brier_score**(*forecast*, *reference*, ***metric_kwargs*)

Brier score of an ensemble for exceeding given thresholds.

$$CRPS = \int_f BS(F(f), H(f-o))df,$$

where $F(o) = \int_{f \leq o} p(f)df$ is the cumulative distribution function (CDF) of the forecast distribution F, o is a point estimate of the true observation (observational error is neglected), BS denotes the Brier score and $H(x)$ denotes the Heaviside step function, which we define here as equal to 1 for $x \geq 0$ and 0 otherwise.

Parameters

- **forecast** (*xr.object*) – Forecast with member dim.
- **reference** (*xr.object*) – References without member dim.
- **threshold** (*int, float, xr.object*) – Threshold to check exceedance, see properscoring.threshold_brier_score.

Details:

minimum	0.0
maximum	1.0
perfect	0.0
orientation	negative

References

- Brier, Glenn W. Verification of forecasts expressed in terms of probability." Monthly Weather Review 78, no. 1 (1950). https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.

See also:

- properscoring.threshold_brier_score
- xskillscore.threshold_brier_score

Example

```
>>> compute_perfect_model(ds, control,
                          metric='threshold_brier_score', threshold=.5)
```

2.7.3 User-defined metrics

You can also construct your own metrics via the *climpred.metrics.Metric* class.

Metric(name, function, positive, . . . [, . . .])	Master class for all metrics.

First, write your own metric function, similar to the existing ones with required arguments `forecast`, `reference`, `dim=None`, and `**metric_kwargs`:

```
from climpred.metrics import Metric

def _my_msle(forecast, reference, dim=None, **metric_kwargs):
    """Mean squared logarithmic error (MSLE).
    https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-
↪model/loss-functions/mean-squared-logarithmic-error."""
    return ( (np.log(forecast + 1) + np.log(reference + 1) ) ** 2).mean(dim)
```

Then initialize this metric function with *climpred.metrics.Metric*:

```
_my_msle = Metric(
    name='my_msle',
    function=_my_msle,
    probabilistic=False,
    positive=False,
    unit_power=0,
    )
```

Finally, compute skill based on your own metric:

```
skill = compute_perfect_model(ds, control, metric=_my_msle)
```

Once you come up with an useful metric for your problem, consider contributing this metric to *climpred*, so all users can benefit from your metric, see contributing.

2.7.4 References

2.8 Comparisons

Forecast skill is always evaluated against a reference for verification. In ESM-based predictions, it is common to compare the ensemble mean forecast against the reference.

In hindcast ensembles *compute_hindcast()*, this ensemble mean forecast (`comparison='e2r'`) is expected to perform better than individual ensemble members (`comparison='m2r'`) as the chaotic component of forecasts is expected to be suppressed by this averaging, while the memory of the system sustains. [Boer2016]

HindcastEnsemble skill is computed by default as the ensemble mean forecast against the reference (`comparison='e2r'`).

In perfect-model frameworks *compute_perfect_model()*, there are even more ways of comparisons. [Seferian2018] shows comparison of the ensemble members against the control run (`comparison='m2c'`) and ensemble members against all other ensemble members (`comparison='m2m'`). Furthermore, using the ensemble mean forecast can be also verified against one control member (`comparison='e2c'`) or all members (`comparison='m2e'`) as done in [Griffies1997].

Perfect-model framework comparison defaults to the ensemble mean forecast verified against each member in turns (`comparison='m2e'`).

These different comparisons demand for a normalization factor to arrive at a normalized skill of 1, when skill saturation is reached (ref: metrics).

While HindcastEnsemble skill is computed over all initializations `init` of the hindcast, the resulting skill is a mean forecast skill over all initializations.

PerfectModelEnsemble skill is computed over a supervector comprised of all initializations and members, which allows the computation of the ACC-based skill [Bushuk2018], but also returns a mean forecast skill over all initializations.

The supervector approach shown in [Bushuk2018] and just calculating a distance-based metric like `rmse` over the member dimension as in [Griffies1997] yield very similar results.

2.8.1 Compute over dimension

The optional argument `dim` defines over which dimension a metric is computed. We can apply a metric over `dim` from `['init', 'member', ['member', 'init']]` in *compute_perfect_model()* and `['init', 'member']` in *compute_hindcast()*. The resulting skill is then reduced by this `dim`. Therefore, applying a metric over `dim='member'` creates a skill for all initializations individually. This can show the initial conditions dependence of skill. Likewise when computing skill over `'init'`, we get skill for each member. This `dim` argument is different from the `comparison` argument which just specifies how `forecast` and `reference` are defined.

However, this above logic applies to deterministic metrics. Probabilistic metrics need to be applied to the `member` dimension and `comparison` from `['m2c', 'm2m']` in *compute_perfect_model()* and `'m2r'` comparison in *compute_hindcast()*. Using a probabilistic metric automatically switches internally to using `dim='member'`.

2.8.2 HindcastEnsemble

```
keyword: 'e2r'
```

_e2r(ds, reference[, stack_dims])	Compare the ensemble mean forecast to a reference in HindcastEnsemble.

keyword: 'm2r'

_m2r(ds, reference[, stack_dims])	Compares each member individually to a reference in HindcastEnsemble.

2.8.3 PerfectModelEnsemble

keyword: 'm2e'

_m2e(ds[, stack_dims])	Compare all members to ensemble mean while leaving out the reference in

keyword: 'm2c'

_m2c(ds[, control_member, stack_dims])	Compare all other members forecasts to control member verification.

keyword: 'm2m'

_m2m(ds[, stack_dims])	Compare all members to all others in turn while leaving out the verification member.

keyword: 'e2c'

_e2c(ds[, control_member, stack_dims])	Compare ensemble mean forecast to control member verification.

2.8.4 User-defined comparisons

You can also construct your own comparisons via the *Comparison* class.

Comparison(name, function, hindcast, ... [, ...])	Master class for all comparisons.

First, write your own comparison function, similar to the existing ones. If a comparison should also be used for probabilistic metrics, use stack_dims to return forecast with member dimension and reference without. For deterministic metric, return forecast and reference with identical dimensions:

```
from climpred.comparisons import Comparison, _drop_members

def _my_m2median_comparison(ds, stack_dims=True):
    """Identical to m2e but median."""
    reference_list = []
    forecast_list = []
    supervector_dim = 'member'
    for m in ds.member.values:
```

(continues on next page)

(continued from previous page)

```
        forecast = _drop_members(ds, rmd_member=[m]).median('member')
        reference = ds.sel(member=m).squeeze()
        forecast_list.append(forecast)
        reference_list.append(reference)
    reference = xr.concat(reference_list, supervector_dim)
    forecast = xr.concat(forecast_list, supervector_dim)
    forecast[supervector_dim] = np.arange(forecast[supervector_dim].size)
    reference[supervector_dim] = np.arange(reference[supervector_dim].size)
    return forecast, reference
```

Then initialize this comparison function with *Comparison*:

```
__my_m2median_comparison = Comparison(
    name='m2me',
    function=_my_m2median_comparison,
    probabilistic=False,
    hindcast=False)
```

Finally, compute skill based on your own comparison:

```
skill = compute_perfect_model(ds, control, metric='rmse', comparison=__my_m2median_
↪comparison)
```

Once you come up with an useful comparison for your problem, consider contributing this comparison to `climpred`, so all users can benefit from your comparison, see contributing.

2.8.5 References

2.9 Prediction Terminology

Terminology is often confusing and highly variable amongst those that make predictions in the geoscience community. Here we define some common terms in climate prediction and how we use them in `climpred`.

2.9.1 Simulation Design

Hindcast Ensemble: `m` ensemble members are initialized from a reference simulation (generally a reconstruction from reanalysis) at `n` initialization dates and integrated for `l` lead years [Boer2016] (*HindcastEnsemble*).

Perfect Model Experiment: `m` ensemble members are initialized from a control simulation at `n` randomly chosen initialization dates and integrated for `l` lead years [Griffies1997] (*PerfectModelEnsemble*).

Reconstruction/Assimilation: A "reconstruction" is a model solution that uses observations in some capacity to approximate historical conditions. This could be done via a forced simulation, such as an OMIP run that uses a dynamical ocean/sea ice core with reanalysis forcing from atmospheric winds. This could also be a fully data assimilative model, which assimilates observations into the model solution.

Uninitialized Ensemble: In this framework, an *uninitialized ensemble* is one that is generated by perturbing initial conditions only at one point in the historical run. These are generated via micro (round-off error perturbations) or macro (starting from completely different restart files) methods. Uninitialized ensembles are used to approximate the magnitude of internal climate variability and to confidently extract the forced response (ensemble mean) in the climate system. In `climpred`, we use uninitialized ensembles as a baseline for how important (reoccurring) initializations are for lending predictability to the system. Some modeling centers (such as NCAR) provide a dynamical uninitialized ensemble (the CESM Large Ensemble) along with their initialized prediction system (the CESM Decadal Prediction

Large Ensemble). If this isn't available, one can approximate the unintiailized response by bootstrapping a control simulation.

2.9.2 Forecast Assessment

Accuracy: The average degree of correspondence between individual pairs of forecasts and observations [Murphy1988]; [Jolliffe2011]. Examples include Mean Absolute Error (MAE) and Mean Square Error (MSE). See metrics.

Association: The overall strength of the relationship between individual pairs of forecasts and observations [Jolliffe2011]. The primary measure of association is the Anomaly Correlation Coefficient (ACC), which can be measured using the Pearson product-moment correlation or Spearman's Rank correlation. See metrics.

(Potential) Predictability: This characterizes the "ability to be predicted" rather than the current "ability to predict." One acquires this by computing a metric (like the anomaly correlation coefficient (ACC)) between the prediction ensemble and a verification member (in a perfect-model setup) or the reconstruction that initialized it (in a hindcast setup) [Meehl2013].

(Prediction) Skill: Skill assesses the ability of the forecasting system to predict the real world, i.e. observations. It must be compared to some "standard of reference" to truly be considered skill, such as climatology of persistence [Murphy1988].

Skill Score: The most generic skill score can be defined as the following [Murphy1988]:

$$S =$$
$$fracA_f - A_rA_p - A_r,$$

where A_f, A_p, and A_r represent the accuracy of the forecast being assessed, the accuracy of a perfect forecast, and the accuracy of the reference forecast (e.g. persistence), respectively [Murphy1985]. Here, S represents the improvement in accuracy of the forecasts over the reference forecasts relative to the total possible improvement in accuracy. They are typically designed to take a value of 1 for a perfect forecast and 0 for equivelant to the reference forecast [Jolliffe2011].

2.9.3 Forecasting

Hindcast: Retrospective forecasts of the past initialized from a reconstruction integrated under external forcing [Boer2016].

Prediction: Forecasts initialized from a reconstruction integrated into the future with external forcing [Boer2016].

Projection An estimate of the future climate that is dependent on the externally forced climate response, such as anthropogenic greenhouse gases, aerosols, and volcanic eruptions [Meehl2013].

2.9.4 References

2.10 Baseline Forecasts

To quantify the quality of an initialized forecast, it is useful to judge it against some simple baseline forecast. `climpred` currently supports a persistence forecast, but future releases will allow computation of other baseline forecasts. Consider opening a Pull Request to get it implemented more quickly.

Persistence Forecast: Whatever is observed at the time of initialization is forecasted to persist into the forecast period [Jolliffe2012]. You can compute this directly via `compute_persistence()` or as a method of `HindcastEnsemble` and `PerfectModelEnsemble`.

Damped Persistence Forecast: (*Not Implemented*) The amplitudes of the anomalies reduce in time exponentially at a time scale of the local autocorrelation [Yuan2016].

$$v_{dp}(t) = v(0)e^{-\alpha t}$$

Climatology: (*Not Implemented*) The average values at the temporal forecast resolution (e.g., annual, monthly) over some long period, which is usually 30 years [Jolliffe2012].

Random Mechanism: (*Not Implemented*) A probability distribution is assigned to the possible range of the variable being forecasted, and a sequence of forecasts is produced by taking a sequence of independent values from that distribution [Jolliffe2012]. This would be similar to computing an uninitialized forecast, using `climpred`'s *`compute_uninitialized()`* function.

2.10.1 References

Help & Reference

- *API Reference*
- *What's New*
- *Helpful Links*
- *Publications Using climpred*
- *Contribution Guide*
- *Release Procedure*
- *Contributors*

2.11 API Reference

This page provides an auto-generated summary of climpred's API. For more details and examples, refer to the relevant chapters in the main part of the documentation.

2.11.1 High-Level Classes

A primary feature of `climpred` is our prediction ensemble objects, *`HindcastEnsemble`* and *`PerfectModelEnsemble`*. Users can append their initialized ensemble to these classes, as well as an arbitrary number of references (assimilations, reconstructions, observations), control runs, and uninitialized ensembles.

HindcastEnsemble

A `HindcastEnsemble` is a prediction ensemble that is initialized off of some form of observations (an assimilation, renanalysis, etc.). Thus, it is anticipated that forecasts are verified against observation-like references. Read more about the terminology here.

`HindcastEnsemble`(xobj)	An object for climate prediction ensembles initialized by a data-like product.

climpred.classes.HindcastEnsemble

class climpred.classes.**HindcastEnsemble**(*xobj*)

An object for climate prediction ensembles initialized by a data-like product.

HindcastEnsemble is a sub-class of *PredictionEnsemble*. It tracks all simulations/observations associated with the prediction ensemble for easy computation across multiple variables and products.

This object is built on *xarray* and thus requires the input object to be an *xarray* Dataset or DataArray.

__init__(*xobj*)

Create a *HindcastEnsemble* object by inputting output from a prediction ensemble in *xarray* format.

Parameters **xobj** (*xarray object*) – decadal prediction ensemble output.

reference

Dictionary of various reference observations/simulations to associate with the decadal prediction ensemble.

uninitialized

Dictionary of companion (or bootstrapped) uninitialized ensemble run.

Methods

__init__(xobj)	Create a *HindcastEnsemble* object by inputting output from a prediction ensemble in *xarray* format.
add_reference(xobj, name)	Add a reference product for comparison to the initialized ensemble.
add_uninitialized(xobj)	Add a companion uninitialized ensemble for comparison to references.
compute_metric([refname, metric, ...])	Compares the initialized ensemble to a given reference.
compute_persistence([refname, metric, max_dof])	Compute a simple persistence forecast for a reference.
compute_uninitialized([refname, metric, ...])	Compares the uninitialized ensemble to a given reference.
get_initialized()	Returns the xarray dataset for the initialized ensemble.
get_reference([name])	Returns the given reference(s).
get_uninitialized()	Returns the xarray dataset for the uninitialized ensemble.
smooth([smooth_kws])	Smooth all entries of PredictionEnsemble in the same manner to be able to still calculate prediction skill afterwards.

Add and Retrieve Datasets

HindcastEnsemble.__init__(xobj)	Create a *HindcastEnsemble* object by inputting output from a prediction ensemble in *xarray* format.
HindcastEnsemble.add_reference(xobj, name)	Add a reference product for comparison to the initialized ensemble.

Continued on next page

Table 11 – continued from previous page

HindcastEnsemble.add_uninitialized(xobj)	Add a companion uninitialized ensemble for comparison to references.
HindcastEnsemble.get_initialized()	Returns the xarray dataset for the initialized ensemble.
HindcastEnsemble.get_reference([name])	Returns the given reference(s).
HindcastEnsemble.get_uninitialized()	Returns the xarray dataset for the uninitialized ensemble.

climpred.classes.HindcastEnsemble.__init__

HindcastEnsemble.**__init__**(*xobj*)

Create a *HindcastEnsemble* object by inputting output from a prediction ensemble in *xarray* format.

Parameters **xobj** (*xarray object*) – decadal prediction ensemble output.

reference

Dictionary of various reference observations/simulations to associate with the decadal prediction ensemble.

climpred.classes.**uninitialized**

Dictionary of companion (or bootstrapped) uninitialized ensemble run.

climpred.classes.HindcastEnsemble.add_reference

HindcastEnsemble.**add_reference**(*xobj*, *name*)

Add a reference product for comparison to the initialized ensemble.

Parameters

- **xobj** (*xarray object*) – Dataset/DataArray being appended to the *HindcastEnsemble* object.
- **name** (*str*) – Name of this object (e.g., "reconstruction")

climpred.classes.HindcastEnsemble.add_uninitialized

HindcastEnsemble.**add_uninitialized**(*xobj*)

Add a companion uninitialized ensemble for comparison to references.

Parameters **xobj** (*xarray object*) – Dataset/DataArray of the uninitialzed ensemble.

climpred.classes.HindcastEnsemble.get_initialized

HindcastEnsemble.**get_initialized**()

Returns the xarray dataset for the initialized ensemble.

climpred.classes.HindcastEnsemble.get_reference

HindcastEnsemble.**get_reference**(*name=None*)

Returns the given reference(s).

Parameters **name** (*str*) – Name of the reference to return (optional)

Returns Dictionary of xarray datasets (if name is None) or single xarray dataset.

climpred.classes.HindcastEnsemble.get_uninitialized

HindcastEnsemble.**get_uninitialized**()
Returns the xarray dataset for the uninitialized ensemble.

Analysis Functions

HindcastEnsemble.compute_metric([refname, ...])	Compares the initialized ensemble to a given reference.
HindcastEnsemble.compute_persistence([...])	Compute a simple persistence forecast for a reference.
HindcastEnsemble.compute_uninitialized([...])	Compares the uninitialized ensemble to a given reference.

climpred.classes.HindcastEnsemble.compute_metric

HindcastEnsemble.**compute_metric**(*refname=None*, *metric='pearson_r'*, *comparison='e2r'*, *max_dof=False*)
Compares the initialized ensemble to a given reference.

This will automatically run the comparison against all shared variables between the initialized ensemble and reference.

Parameters

- **refname** (*str*) – Name of reference to compare to. If *None*, compare to all references.
- **metric** (*str, default 'pearson_r'*) – Metric to apply in the comparison.
- **comparison** (*str, default 'e2r'*) – How to compare to the reference. ('e2r' for ensemble mean to reference. 'm2r' for each individual member to reference)
- **max_dof** (*bool, default False*) – If True, maximize the degrees of freedom for each lag calculation.

Returns Dataset of comparison results (if comparing to one reference), or dictionary of Datasets with keys corresponding to reference name.

climpred.classes.HindcastEnsemble.compute_persistence

HindcastEnsemble.**compute_persistence**(*refname=None*, *metric='pearson_r'*, *max_dof=False*)
Compute a simple persistence forecast for a reference.

This simply applies some metric between the reference and itself out to some lag (i.e., an ACF in the case of pearson r).

Parameters

- **refname** (*str, default None*) – Name of reference to compute the persistence forecast for. If *None*, compute for all references.
- **metric** (*str, default 'pearson_r'*) – Metric to apply to the persistence forecast.
- **max_dof** (*bool, default False*) – If True, maximize the degrees of freedom for each lag calculation.

Returns Dataset of persistence forecast results (if refname is declared), or dictionary of Datasets with keys corresponding to reference name.

Reference:

- Chapter 8 (Short-Term Climate Prediction) in Van den Dool, Huug. Empirical methods in short-term climate prediction. Oxford University Press, 2007.

climpred.classes.HindcastEnsemble.compute_uninitialized

HindcastEnsemble.**compute_uninitialized**(*refname=None*, *metric='pearson_r'*, *comparison='e2r'*)

Compares the uninitialized ensemble to a given reference.

This will automatically run the comparison against all shared variables between the initialized ensemble and reference.

Parameters

- **refname** (*str*) – Name of reference to compare to. If *None*, compare to all references.
- **metric** (*str, default 'pearson_r'*) – Metric to apply in the comparison.
- **comparison** (*str, default 'e2r'*) – How to compare to the reference. ('e2r' for ensemble mean to reference. 'm2r' for each individual member to reference)

Returns Dataset of comparison results (if comparing to one reference), or dictionary of Datasets with keys corresponding to reference name.

Pre-Processing

HindcastEnsemble.smooth([smooth_kws])	Smooth all entries of PredictionEnsemble in the same manner to be able to still calculate prediction skill afterwards.

climpred.classes.HindcastEnsemble.smooth

HindcastEnsemble.**smooth**(*smooth_kws='goddard2013'*)

Smooth all entries of PredictionEnsemble in the same manner to be able to still calculate prediction skill afterwards.

Parameters **xobj** (*xarray object*) – decadal prediction ensemble output.

smooth_kws

Dictionary to specify the dims to smooth compatible with *spatial_smoothing_xesmf*, *temporal_smoothing* or *spatial_smoothing_xrcoarsen*. Shortcut for Goddard et al. 2013 recommendations: 'goddard2013'

Type dict or str

Example: >>> PredictionEnsemble.smooth(smooth_kws={'time': 2,

'lat': 5, 'lon': 4'})

```
>>> PredictionEnsemble.smooth(smooth_kws='goddard2013')
```

PerfectModelEnsemble

A `PerfectModelEnsemble` is a prediction ensemble that is initialized off of a control simulation for a number of randomly chosen initialization dates. Thus, forecasts cannot be verified against real-world observations. Instead, they are compared to one another and to the original control run. Read more about the terminology here.

PerfectModelEnsemble(xobj)	An object for "perfect model" climate prediction ensembles.

climpred.classes.PerfectModelEnsemble

class climpred.classes.**PerfectModelEnsemble**(*xobj*)

An object for "perfect model" climate prediction ensembles.

PerfectModelEnsemble is a sub-class of *PredictionEnsemble*. It tracks the control run used to initialize the ensemble for easy computations, bootstrapping, etc.

This object is built on *xarray* and thus requires the input object to be an *xarray* Dataset or DataArray.

__init__(*xobj*)

Create a *PerfectModelEnsemble* object by inputting output from the control run in *xarray* format.

Parameters **xobj** (*xarray object*) – decadal prediction ensemble output.

control

Dictionary of control run associated with the initialized ensemble.

uninitialized

Dictionary of uninitialized run that is bootstrapped from the initialized run.

Methods

__init__(xobj)	Create a *PerfectModelEnsemble* object by inputting output from the control run in *xarray* format.
add_control(xobj)	Add the control run that initialized the climate prediction ensemble.
bootstrap([metric, comparison, sig, ...])	Bootstrap ensemble simulations with replacement.
compute_metric([metric, comparison])	Compares the initialized ensemble to the control run.
compute_persistence([metric])	Compute a simple persistence forecast for the control run.
compute_uninitialized([metric, comparison])	Compares the bootstrapped uninitialized run to the control run.
generate_uninitialized()	Generate an uninitialized ensemble by bootstrapping the initialized prediction ensemble.
get_control()	Returns the control as an xarray dataset.
get_initialized()	Returns the xarray dataset for the initialized ensemble.
get_uninitialized()	Returns the xarray dataset for the uninitialized ensemble.
smooth([smooth_kws])	Smooth all entries of PredictionEnsemble in the same manner to be able to still calculate prediction skill afterwards.

Add and Retrieve Datasets

PerfectModelEnsemble.__init__(xobj)	Create a *PerfectModelEnsemble* object by inputting output from the control run in *xarray* format.
PerfectModelEnsemble.add_control(xobj)	Add the control run that initialized the climate prediction ensemble.
PerfectModelEnsemble.get_initialized()	Returns the xarray dataset for the initialized ensemble.
PerfectModelEnsemble.get_control()	Returns the control as an xarray dataset.
PerfectModelEnsemble.get_uninitialized()	Returns the xarray dataset for the uninitialized ensemble.

climpred.classes.PerfectModelEnsemble.__init__

PerfectModelEnsemble.**__init__**(*xobj*)

Create a *PerfectModelEnsemble* object by inputting output from the control run in *xarray* format.

Parameters **xobj** (*xarray object*) – decadal prediction ensemble output.

control

Dictionary of control run associated with the initialized ensemble.

climpred.classes.**uninitialized**

Dictionary of uninitialized run that is bootstrapped from the initialized run.

climpred.classes.PerfectModelEnsemble.add_control

PerfectModelEnsemble.**add_control**(*xobj*)

Add the control run that initialized the climate prediction ensemble.

Parameters **xobj** (*xarray object*) – Dataset/DataArray of the control run.

climpred.classes.PerfectModelEnsemble.get_initialized

PerfectModelEnsemble.**get_initialized**()

Returns the xarray dataset for the initialized ensemble.

climpred.classes.PerfectModelEnsemble.get_control

PerfectModelEnsemble.**get_control**()

Returns the control as an xarray dataset.

climpred.classes.PerfectModelEnsemble.get_uninitialized

PerfectModelEnsemble.**get_uninitialized**()

Returns the xarray dataset for the uninitialized ensemble.

Analysis Functions

PerfectModelEnsemble.bootstrap([metric, ...])	Bootstrap ensemble simulations with replacement.
PerfectModelEnsemble.compute_metric([...])	Compares the initialized ensemble to the control run.
PerfectModelEnsemble.compute_persistence([...])	Compute a simple persistence forecast for the control run.
PerfectModelEnsemble.compute_uninitialized([...])	Compares the bootstrapped uninitialized run to the control run.

climpred.classes.PerfectModelEnsemble.bootstrap

PerfectModelEnsemble.**bootstrap**(*metric='pearson_r'*, *comparison='m2e'*, *sig=95*, *bootstrap=500*, *pers_sig=None*)

Bootstrap ensemble simulations with replacement.

Parameters

- **metric** (*str, default 'pearson_r'*) – Metric to apply for bootstrapping.
- **comparison** (*str, default 'm2e'*) – Comparison style for bootstrapping.
- **sig** (*int, default 95*) – Significance level for uninitialized and initialized comparison.
- **bootstrap** (*int, default 500*) – Number of resampling iterations for bootstrapping with replacement.
- **pers_sig** (*int, default None*) – If not None, the separate significance level for persistence.

Returns

Dictionary of Datasets for each variable applied to with the following variables:

- init_ci: confidence levels of init_skill.
- uninit_ci: confidence levels of uninit_skill.
- pers_ci: confidence levels of pers_skill.
- **p_uninit_over_init: p value of the hypothesis that the** difference of skill between the initialized and uninitialized simulations is smaller or equal to zero based on bootstrapping with replacement.
- **p_pers_over_init: p value of the hypothesis that the** difference of skill between the initialized and persistence simulations is smaller or equal to zero based on bootstrapping with replacement.

Reference:

- Goddard, L., A. Kumar, A. Solomon, D. Smith, G. Boer, P. Gonzalez, V. Kharin, et al. "A Verification Framework for Interannual-to-Decadal Predictions Experiments." Climate Dynamics 40, no. 1–2 (January 1, 2013): 245–72. https://doi.org/10/f4jjvf.

climpred.classes.PerfectModelEnsemble.compute_metric

PerfectModelEnsemble.**compute_metric**(*metric='pearson_r'*, *comparison='m2m'*)

Compares the initialized ensemble to the control run.

Parameters

- **metric** (*str, default 'pearson_r'*) – Metric to apply in the comparison.
- **comparison** (*str, default 'm2m'*) – How to compare the climate prediction ensemble to the control.

Returns Result of the comparison as a Dataset.

climpred.classes.PerfectModelEnsemble.compute_persistence

PerfectModelEnsemble.**compute_persistence**(*metric='pearson_r'*)

Compute a simple persistence forecast for the control run.

Parameters **metric** (*str, default 'pearson_r'*) – Metric to apply to the persistence forecast.

Returns Dataset of persistence forecast results (if refname is declared), or dictionary of Datasets with keys corresponding to reference name.

Reference:

- Chapter 8 (Short-Term Climate Prediction) in Van den Dool, Huug. Empirical methods in short-term climate prediction. Oxford University Press, 2007.

climpred.classes.PerfectModelEnsemble.compute_uninitialized

PerfectModelEnsemble.**compute_uninitialized**(*metric='pearson_r', comparison='m2e'*)

Compares the bootstrapped uninitialized run to the control run.

Parameters

- **metric** (*str, default 'pearson_r'*) – Metric to apply in the comparison.
- **comparison** (*str, default 'm2m'*) – How to compare to the control run.
- **running** (*int, default None*) – Size of the running window for variance smoothing.

Returns Result of the comparison as a Dataset.

Generate Data

PerfectModelEnsemble.generate_uninitialized()	Generate an uninitialized ensemble by bootstrapping the initialized prediction ensemble.

climpred.classes.PerfectModelEnsemble.generate_uninitialized

PerfectModelEnsemble.**generate_uninitialized**()

Generate an uninitialized ensemble by bootstrapping the initialized prediction ensemble.

Returns Bootstrapped (uninitialized) ensemble as a Dataset.

2.11.2 Direct Function Calls

A user can directly call functions in `climpred`. This requires entering more arguments, e.g. the initialized ensemble `Dataset`/`xarray.core.dataarray.DataArray` directly as well as a reference product or control run. Our object *HindcastEnsemble* and *PerfectModelEnsemble* wrap most of these functions, making the analysis process much simpler. Once we have wrapped all of the functions in their entirety, we will likely depricate the ability to call them directly.

Bootstrap

bootstrap_compute(hind, reference[, hist, ...])	Bootstrap compute with replacement.
bootstrap_hindcast(hind, hist, reference[, ...])	Bootstrap compute with replacement. Wrapper of
bootstrap_perfect_model(ds, control[, ...])	Bootstrap compute with replacement. Wrapper of
bootstrap_uninit_pm_ensemble_from_control(ds, ...)	Create a pseudo-ensemble from control run.
bootstrap_uninitialized_ensemble(hind, hist)	Resample uninitialized hindcast from historical members.
dpp_threshold(control[, sig, bootstrap, dim])	Calc DPP significance levels from re-sampled dataset.
varweighted_mean_period_threshold(control[, ...])	Calc the variance-weighted mean period significance levels from re-sampled dataset.

climpred.bootstrap.bootstrap_compute

climpred.bootstrap.**bootstrap_compute**(*hind*, *reference*, *hist=None*, *metric='pearson_r'*, *comparison='m2e'*, *dim='init'*, *sig=95*, *bootstrap=500*, *pers_sig=None*, *compute=<function compute_hindcast>*, *resample_uninit=<function bootstrap_uninitialized_ensemble>*, ***metric_kwargs*)

Bootstrap compute with replacement.

Parameters

- **hind** (*xr.Dataset*) – prediction ensemble.
- **reference** (*xr.Dataset*) – reference simulation.
- **hist** (*xr.Dataset*) – historical/uninitialized simulation.
- **metric** (*str*) – *metric*. Defaults to 'pearson_r'.
- **comparison** (*str*) – *comparison*. Defaults to 'm2e'.
- **dim** (*str or list*) – dimension to apply metric over. default: 'init'
- **sig** (*int*) – Significance level for uninitialized and initialized skill. Defaults to 95.
- **pers_sig** (*int*) – Significance level for persistence skill confidence levels. Defaults to sig.
- **bootstrap** (*int*) – number of resampling iterations (bootstrap with replacement). Defaults to 500.
- **compute** (*func*) – function to compute skill. Choose from [*climpred.prediction.compute_perfect_model()*, *climpred.prediction.compute_hindcast()*].

- **resample_uninit** (*func*) – function to create an uninitialized ensemble from a control simulation or uninitialized large ensemble. Choose from: [*bootstrap_uninitialized_ensemble()*, *bootstrap_uninit_pm_ensemble_from_control()*].
- **metric_kwargs** (****) – additional keywords to be passed to metric (see the arguments required for a given metric in *Metrics*).

Returns

(xr.Dataset): bootstrapped results

- init_ci (xr.Dataset): confidence levels of init_skill
- uninit_ci (xr.Dataset): confidence levels of uninit_skill
- **p_uninit_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and uninitialized simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.
- pers_ci (xr.Dataset): confidence levels of pers_skill
- **p_pers_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and persistence simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.

Return type results

Reference:

- Goddard, L., A. Kumar, A. Solomon, D. Smith, G. Boer, P. Gonzalez, V. Kharin, et al. "A Verification Framework for Interannual-to-Decadal Predictions Experiments." Climate Dynamics 40, no. 1–2 (January 1, 2013): 245–72. https://doi.org/10/f4jjvf.

See also:

- climpred.bootstrap.bootstrap_hindcast
- climpred.bootstrap.bootstrap_perfect_model

climpred.bootstrap.bootstrap_hindcast

climpred.bootstrap.**bootstrap_hindcast**(*hind*, *hist*, *reference*, *metric='pearson_r'*, *comparison='e2r'*, *dim='init'*, *sig=95*, *bootstrap=500*, *pers_sig=None*, ***metric_kwargs*)

Bootstrap compute with replacement. Wrapper of py:func:*bootstrap_compute* for hindcasts.

Parameters

- **hind** (*xr.Dataset*) – prediction ensemble.
- **reference** (*xr.Dataset*) – reference simulation.
- **hist** (*xr.Dataset*) – historical/uninitialized simulation.
- **metric** (*str*) – *metric*. Defaults to 'pearson_r'.
- **comparison** (*str*) – *comparison*. Defaults to 'e2r'.
- **dim** (*str*) – dimension to apply metric over. default: 'init'

- **sig** (*int*) – Significance level for uninitialized and initialized skill. Defaults to 95.
- **pers_sig** (*int*) – Significance level for persistence skill confidence levels. Defaults to sig.
- **bootstrap** (*int*) – number of resampling iterations (bootstrap with replacement). Defaults to 500.
- **metric_kwargs** (****) – additional keywords to be passed to metric (see the arguments required for a given metric in *Metrics*).

Returns

(xr.Dataset): bootstrapped results

- init_ci (xr.Dataset): confidence levels of init_skill
- uninit_ci (xr.Dataset): confidence levels of uninit_skill
- **p_uninit_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and uninitialized simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.
- pers_ci (xr.Dataset): confidence levels of pers_skill
- **p_pers_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and persistence simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.

Return type results

Reference:

- Goddard, L., A. Kumar, A. Solomon, D. Smith, G. Boer, P. Gonzalez, V. Kharin, et al. "A Verification Framework for Interannual-to-Decadal Predictions Experiments." Climate Dynamics 40, no. 1–2 (January 1, 2013): 245–72. https://doi.org/10/f4jjvf.

See also:

- climpred.bootstrap.bootstrap_compute
- climpred.prediction.compute_hindcast

climpred.bootstrap.bootstrap_perfect_model

climpred.bootstrap.**bootstrap_perfect_model**(*ds*, *control*, *metric='pearson_r'*, *comparison='m2e'*, *dim=None*, *sig=95*, *bootstrap=500*, *pers_sig=None*, ***metric_kwargs*)

Bootstrap compute with replacement. Wrapper of py:func:*bootstrap_compute* for perfect-model framework.

Parameters

- **hind** (*xr.Dataset*) – prediction ensemble.
- **reference** (*xr.Dataset*) – reference simulation.
- **hist** (*xr.Dataset*) – historical/uninitialized simulation.
- **metric** (*str*) – *metric*. Defaults to 'pearson_r'.

- **comparison** (*str*) – *comparison*. Defaults to 'm2e'.
- **dim** (*str*) – dimension to apply metric over. default: ['init', 'member']
- **sig** (*int*) – Significance level for uninitialized and initialized skill. Defaults to 95.
- **pers_sig** (*int*) – Significance level for persistence skill confidence levels. Defaults to sig.
- **bootstrap** (*int*) – number of resampling iterations (bootstrap with replacement). Defaults to 500.
- **metric_kwargs** (****) – additional keywords to be passed to metric (see the arguments required for a given metric in *Metrics*).

Returns

(xr.Dataset): bootstrapped results

- init_ci (xr.Dataset): confidence levels of init_skill
- uninit_ci (xr.Dataset): confidence levels of uninit_skill
- **p_uninit_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and uninitialized simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.
- pers_ci (xr.Dataset): confidence levels of pers_skill
- **p_pers_over_init (xr.Dataset): p value of the hypothesis** that the difference of skill between the initialized and persistence simulations is smaller or equal to zero based on bootstrapping with replacement. Defaults to None.

Return type results

Reference:

- Goddard, L., A. Kumar, A. Solomon, D. Smith, G. Boer, P. Gonzalez, V. Kharin, et al. "A Verification Framework for Interannual-to-Decadal Predictions Experiments." Climate Dynamics 40, no. 1–2 (January 1, 2013): 245–72. https://doi.org/10/f4jjvf.

See also:

- climpred.bootstrap.bootstrap_compute
- climpred.prediction.compute_perfect_model

climpred.bootstrap.bootstrap_uninit_pm_ensemble_from_control

climpred.bootstrap.**bootstrap_uninit_pm_ensemble_from_control**(*ds*, *control*)

Create a pseudo-ensemble from control run.

Note: Needed for block bootstrapping confidence intervals of a metric in perfect model framework. Takes randomly segments of length of ensemble dataset from control and rearranges them into ensemble and member dimensions.

Parameters

- **ds** (*xarray object*) – ensemble simulation.

- **control** (*xarray object*) – control simulation.

Returns pseudo-ensemble generated from control run.

Return type ds_e (xarray object)

climpred.bootstrap.bootstrap_uninitialized_ensemble

climpred.bootstrap.**bootstrap_uninitialized_ensemble**(*hind*, *hist*)

Resample uninitialized hindcast from historical members.

Note: Needed for bootstrapping confidence intervals and p_values of a metric in the hindcast framework. Takes hind.lead.size timesteps from historical at same forcing and rearranges them into ensemble and member dimensions.

Parameters

- **hind** (*xarray object*) – hindcast.
- **hist** (*xarray object*) – historical uninitialized.

Returns uninitialize hindcast with hind.coords.

Return type uninit_hind (xarray object)

climpred.bootstrap.dpp_threshold

climpred.bootstrap.**dpp_threshold**(*control*, *sig=95*, *bootstrap=500*, *dim='time'*, ***dpp_kwargs*)

Calc DPP significance levels from re-sampled dataset.

Reference:

- Feng, X., T. DelSole, and P. Houser. "Bootstrap Estimated Seasonal Potential Predictability of Global Temperature and Precipitation." Geophysical Research Letters 38, no. 7 (2011). https://doi.org/10/ft272w.

See also:

- climpred.bootstrap._bootstrap_func
- climpred.stats.dpp

climpred.bootstrap.varweighted_mean_period_threshold

climpred.bootstrap.**varweighted_mean_period_threshold**(*control*, *sig=95*, *bootstrap=500*, *time_dim='time'*)

Calc the variance-weighted mean period significance levels from re-sampled dataset.

See also:

- climpred.bootstrap._bootstrap_func
- climpred.stats.varweighted_mean_period

Prediction

compute_hindcast(hind, reference[, metric, ...])	Compute a predictability skill score against a reference
compute_perfect_model(ds, control[, metric, ...])	Compute a predictability skill score for a perfect-model framework simulation dataset.
compute_persistence(hind, reference[, ...])	Computes the skill of a persistence forecast from a simulation.
compute_uninitialized(uninit, reference[, ...])	Compute a predictability score between an uninitialized ensemble and a reference.

climpred.prediction.compute_hindcast

climpred.prediction.**compute_hindcast**(*hind*, *reference*, *metric='pearson_r'*, *comparison='e2r'*, *dim='init'*, *max_dof=False*, *add_attrs=True*, ***metric_kwargs*)

Compute a predictability skill score against a reference

Parameters

- **hind** (*xarray object*) – Expected to follow package conventions: * `init` : dim of initialization dates * `lead` : dim of lead time from those initializations Additional dims can be member, lat, lon, depth, ...
- **reference** (*xarray object*) – reference output/data over same time period.
- **metric** (*str*) – Metric used in comparing the decadal prediction ensemble with the reference (see `climpred.utils.get_metric_class()` and *Metrics*).
- **comparison** (*str*) – How to compare the decadal prediction ensemble to the reference:
  - e2r : ensemble mean to reference (Default)
  - m2r : each member to the reference

  (see *Comparisons*)
- **dim** (*str or list*) – dimension to apply metric over. default: 'init'
- **max_dof** (*bool*) – If True, maximize the degrees of freedom by slicing *hind* and *reference* to a common time frame at each lead.

  If False (default), then slice to a common time frame prior to computing metric. This philosophy follows the thought that each lead should be based on the same set of initializations.
- **add_attrs** (*bool*) – write climpred compute args to attrs. default: True
- **metric_kwargs** (****) – additional keywords to be passed to metric (see the arguments required for a given metric in *Metrics*).

Returns Predictability with main dimension `lag` without dimension `dim`

Return type skill (xarray object)

climpred.prediction.compute_perfect_model

climpred.prediction.**compute_perfect_model**(*ds*, *control*, *metric='pearson_r'*, *comparison='m2e'*, *dim=None*, *add_attrs=True*, ***metric_kwargs*)

Compute a predictability skill score for a perfect-model framework simulation dataset.

Parameters

- **ds** (*xarray object*) – ensemble with dims `lead`, `init`, `member`.
- **control** (*xarray object*) – control with dimension `time`.
- **metric** (*str*) – *metric* name, see `climpred.utils.get_metric_class()` and (see *Metrics*).
- **comparison** (*str*) – *comparison* name defines what to take as forecast and verification (see `climpred.utils.get_comparison_class()` and *Comparisons*).
- **dim** (*str or list*) – dimension to apply metric over. default: ['member', 'init']
- **add_attrs** (*bool*) – write climpred compute args to attrs. default: True
- **metric_kwargs** (****) – additional keywords to be passed to metric. (see the arguments required for a given metric in metrics.py)

Returns

skill score with dimensions as input *ds* without *dim*.

Return type skill (xarray object)

climpred.prediction.compute_persistence

climpred.prediction.**compute_persistence**(*hind*, *reference*, *metric='pearson_r'*, *max_dof=False*, ***metric_kwargs*)

Computes the skill of a persistence forecast from a simulation.

Parameters

- **hind** (*xarray object*) – The initialized ensemble.
- **reference** (*xarray object*) – The reference time series.
- **metric** (*str*) – Metric name to apply at each lag for the persistence computation. Default: 'pearson_r'
- **max_dof** (*bool*) – If True, maximize the degrees of freedom by slicing *hind* and *reference* to a common time frame at each lead.

  If False (default), then slice to a common time frame prior to computing metric. This philosophy follows the thought that each lead should be based on the same set of initializations.
- **metric_kwargs** (****) – additional keywords to be passed to metric (see the arguments required for a given metric in *Metrics*).

Returns Results of persistence forecast with the input metric applied.

Return type pers (xarray object)

Reference:

- Chapter 8 (Short-Term Climate Prediction) in Van den Dool, Huug. Empirical methods in short-term climate prediction. Oxford University Press, 2007.

climpred.prediction.compute_uninitialized

climpred.prediction.**compute_uninitialized**(*uninit*, *reference*, *metric='pearson_r'*, *comparison='e2r'*, *dim='time'*, *add_attrs=True*, ***metric_kwargs*)

Compute a predictability score between an uninitialized ensemble and a reference.

Note: Based on Decadal Prediction protocol, this should only be computed for the first lag and then projected out to any further lags being analyzed.

Parameters

- **uninit** (*xarray object*) – uninitialized ensemble.
- **reference** (*xarray object*) – reference output/data over same time period.
- **metric** (*str*) – Metric used in comparing the uninitialized ensemble with the reference.
- **comparison** (*str*) –

  How to compare the uninitialized ensemble to the reference:

  - e2r : ensemble mean to reference (Default)
  - m2r : each member to the reference
- **add_attrs** (*bool*) – write climpred compute args to attrs. default: True
- **metric_kwargs** (****) – additional keywords to be passed to metric

Returns Results from comparison at the first lag.

Return type u (xarray object)

Metrics

Metric(name, function, positive, . . . [, . . .])	Master class for all metrics.
_get_norm_factor(comparison)	Get normalization factor for normalizing distance metrics.

climpred.metrics.Metric

class climpred.metrics.**Metric**(*name*, *function*, *positive*, *probabilistic*, *unit_power*, *long_name=None*, *aliases=None*, *minimum=None*, *maximum=None*, *perfect=None*)

Master class for all metrics.

__init__(*name*, *function*, *positive*, *probabilistic*, *unit_power*, *long_name=None*, *aliases=None*, *minimum=None*, *maximum=None*, *perfect=None*)

Metric initialization.

Parameters

- **name** (*str*) – name of metric.
- **function** (*function*) – metric function.
- **positive** (*bool*) – Is metric positively oriented? Higher metric values means higher skill.

- **probabilistic** (*bool*) – Is metric probabilistic? *False* means deterministic.
- **unit_power** (*float, int*) – Power of the unit of skill based on unit of input, e.g. input unit [m]: skill unit [(m)**unit_power]
- **long_name** (*str, optional*) – long_name of metric. Defaults to None.
- **aliases** (*list of str, optional*) – Allowed aliases for this metric. Defaults to None.
- **min** (*float, optional*) – Minimum skill for metric. Defaults to None.
- **max** (*float, optional*) – Maxmimum skill for metric. Defaults to None.
- **perfect** (*float, optional*) – Perfect skill for metric. Defaults to None.

Returns metric class Metric.

Return type *Metric*

Methods

__init__(name, function, positive, ... [, ...])	Metric initialization.

climpred.metrics._get_norm_factor

climpred.metrics.**_get_norm_factor**(*comparison*)

Get normalization factor for normalizing distance metrics.

A distance metric is normalized by the standard deviation or variance of a reference/control simulation. The goal of a normalized distance metric is to get a constant and comparable value of typically 1 (or 0 for metrics defined as 1 - metric), when the metric saturates and the predictability horizon is reached.

To directly compare skill between different comparisons used, a factor is added in the normalized metric formula, see Seferian et al. 2018. For example, NRMSE gets smaller in comparison `m2e` than `m2m` by design, because the ensemble mean is always closer to individual ensemble members than ensemble members to each other.

Note: This is used for NMSE, NRMSE, MSSS, NMAE.

Parameters **comparison** (*class*) – comparison class.

Returns normalization factor.

Return type fac (int)

Raises **KeyError** – if comparison is not matching.

Example

```
>>> # check skill saturation value of roughly 1 for different comparisons
>>> metric = 'nrmse'
>>> for c in ['m2m', 'm2e', 'm2c', 'e2c']:
        s = compute_perfect_model(ds, control, metric=metric, comparison=c)
        s.plot(label=' '.join([metric,c]))
>>> plt.legend()
```

Reference:

- Séférian, Roland, Sarah Berthet, and Matthieu Chevallier. "Assessing the Decadal Predictability of Land and Ocean Carbon Uptake." Geophysical Research Letters, March 15, 2018. https://doi.org/10/gdb424.

Comparisons

Comparison(name, function, hindcast, ... [, ...])	Master class for all comparisons.

climpred.comparisons.Comparison

class climpred.comparisons.**Comparison**(*name*, *function*, *hindcast*, *probabilistic*, *long_name=None*)

Master class for all comparisons.

__init__(*name*, *function*, *hindcast*, *probabilistic*, *long_name=None*)

Comparison initialization.

Parameters

- **name** (*str*) – name of comparison.
- **function** (*function*) – comparison function.
- **hindcast** (*bool*) – Can comparison be used in *compute_hindcast*? *False* means *compute_perfect_model*
- **probabilistic** (*bool*) – Can this comparison be used for probabilistic metrics also? Probabilistic metrics require multiple forecasts. *False* means that comparison is only deterministic. *True* means that comparison can be used both deterministic and probabilistic.
- **long_name** (*str, optional*) – longname of comparison. Defaults to None.

Returns comparison class Comparison.

Return type comparison

Methods

__init__(name, function, hindcast, probabilistic)	Comparison initialization.

Statistics

autocorr(ds[, lag, dim, return_p])	Calculate the lagged correlation of time series.
corr(x, y[, dim, lag, return_p])	Computes the Pearson product-moment coefficient of linear correlation.
decorrelation_time(da[, r, dim])	Calculate the decorrelaton time of a time series.
dpp(ds[, dim, m, chunk])	Calculates the Diagnostic Potential Predictability (dpp)
rm_poly(ds, order[, dim])	Returns xarray object with nth-order fit removed.
rm_trend(da[, dim])	Remove linear trend from time series.
varweighted_mean_period(da[, dim])	Calculate the variance weighted mean period of time series based on xrft.power_spectrum.

climpred.stats.autocorr

climpred.stats.**autocorr**(*ds*, *lag=1*, *dim='time'*, *return_p=False*)

Calculate the lagged correlation of time series.

Parameters

- **ds** (*xarray object*) – Time series or grid of time series.
- **lag** (*optional int*) – Number of time steps to lag correlate to.
- **dim** (*optional str*) – Name of dimension to autocorrelate over.
- **return_p** (*optional bool*) – If True, return correlation coefficients and p values.

Returns

Pearson correlation coefficients.

If return_p, also returns their associated p values.

climpred.stats.corr

climpred.stats.**corr**(*x*, *y*, *dim='time'*, *lag=0*, *return_p=False*)

Computes the Pearson product-moment coefficient of linear correlation.

Note: This version calculates the effective degrees of freedom, accounting for autocorrelation within each time series that could fluff the significance of the correlation.

Parameters

- **x** (*xarray object*) – Independent variable time series or grid of time series.
- **y** (*xarray object*) – Dependent variable time series or grid of time series
- **dim** (*optional str*) – Correlation dimension
- **lag** (*optional int*) – Lag to apply to correlaton, with x predicting y.
- **return_p** (*optional bool*) – If True, return correlation coefficients as well as p values.

Returns Pearson correlation coefficients If return_p True, associated p values.

References

- Wilks, Daniel S. Statistical methods in the atmospheric sciences. Vol. 100. Academic press, 2011.
- Lovenduski, Nicole S., and Nicolas Gruber. "Impact of the Southern Annular Mode on Southern Ocean circulation and biology." Geophysical Research Letters 32.11 (2005).

climpred.stats.decorrelation_time

climpred.stats.**decorrelation_time**(*da*, *r=20*, *dim='time'*)

Calculate the decorrelaton time of a time series.

$$\tau_d = 1 + 2 * \sum_{k=1}^{r} (\alpha_k)^k$$

Parameters

- **da** (*xarray object*) – Time series.
- **r** (*optional int*) – Number of iterations to run the above formula.
- **dim** (*optional str*) – Time dimension for xarray object.

Returns Decorrelation time of time series.

Reference:

- Storch, H. v, and Francis W. Zwiers. Statistical Analysis in Climate Research. Cambridge; New York: Cambridge University Press, 1999., p.373

climpred.stats.dpp

climpred.stats.**dpp**(*ds*, *dim='time'*, *m=10*, *chunk=True*)

Calculates the Diagnostic Potential Predictability (dpp)

$$DPP_{\text{unbiased}}(m) = \frac{\sigma_m^2 - \frac{1}{m} \cdot \sigma^2}{\sigma^2}$$

Note: Resplandy et al. 2015 and Seferian et al. 2018 calculate unbiased DPP in a slightly different way: chunk=False.

Parameters

- **ds** (*xr.DataArray*) – control simulation with time dimension as years.
- **dim** (*str*) – dimension to apply DPP on. Default: time.
- **m** (*optional int*) – separation time scale in years between predictable low-freq component and high-freq noise.
- **chunk** (*optional boolean*) – Whether chunking is applied. Default: True. If False, then uses Resplandy 2015 / Seferian 2018 method.

Returns ds without time dimension.

Return type dpp (xr.DataArray)

References

- Boer, G. J. "Long Time-Scale Potential Predictability in an Ensemble of Coupled Climate Models." Climate Dynamics 23, no. 1 (August 1, 2004): 29–44. https://doi.org/10/csjjbh.

- Resplandy, L., R. Séférian, and L. Bopp. "Natural Variability of CO2 and O2 Fluxes: What Can We Learn from Centuries-Long Climate Models Simulations?" Journal of Geophysical Research: Oceans 120, no. 1 (January 2015): 384–404. https://doi.org/10/f63c3h.
- Séférian, Roland, Sarah Berthet, and Matthieu Chevallier. "Assessing the Decadal Predictability of Land and Ocean Carbon Uptake." Geophysical Research Letters, March 15, 2018. https://doi.org/10/gdb424.

climpred.stats.rm_poly

climpred.stats.**rm_poly**(*ds*, *order*, *dim='time'*)

Returns xarray object with nth-order fit removed.

Note: This automatically performs a linear interpolation across any NaNs in the time series.

Parameters

- **ds** (*xarray object*) – Time series to be detrended.
- **order** (*int*) – Order of polynomial fit to be removed.
- **dim** (*optional str*) – Dimension over which to remove the polynomial fit.

Returns xarray object with polynomial fit removed.

climpred.stats.rm_trend

climpred.stats.**rm_trend**(*da*, *dim='time'*)

Remove linear trend from time series.

Parameters

- **ds** (*xarray object*) – Time series to be detrended.
- **dim** (*optional str*) – Dimension over which to remove the linear trend.

Returns xarray object with linear trend removed.

climpred.stats.varweighted_mean_period

climpred.stats.**varweighted_mean_period**(*da*, *dim='time'*, ***kwargs*)

Calculate the variance weighted mean period of time series based on xrft.power_spectrum.

$$P_x = \frac{\sum_k V(f_k, x)}{\sum_k f_k \cdot V(f_k, x)}$$

Parameters

- **da** (*xarray object*) – input data including dim.
- **dim** (*optional str*) – Name of time dimension.
- ****kwargs see xrft.power_spectrum** (*for*) –

Reference:

- Branstator, Grant, and Haiyan Teng. "Two Limits of Initial-Value Decadal Predictability in a CGCM." Journal of Climate 23, no. 23 (August 27, 2010): 6292-6311. https://doi.org/10/bwq92h.

See also: https://xrft.readthedocs.io/en/latest/api.html#xrft.xrft.power_spectrum

Tutorial

load_dataset([name, cache, cache_dir, ...])	Load example data or a mask from an online repository.

climpred.tutorial.load_dataset

climpred.tutorial.**load_dataset**(*name=None*, *cache=True*, *cache_dir='~/.climpred_data'*, *github_url='https://github.com/bradyrx/climpred-data'*, *branch='master'*, *extension=None*, *proxy_dict=None*, ***kws*)

Load example data or a mask from an online repository.

Parameters

- **name** – (str, default None) Name of the netcdf file containing the dataset, without the .nc extension. If None, this function prints out the available datasets to import.
- **cache_dir** – (str, optional) The directory in which to search for and cache the data.
- **cache** – (bool, optional) If True, cache data locally for use on later calls.
- **github_url** – (str, optional) Github repository where the data is stored.
- **branch** – (str, optional) The git branch to download from.
- **extension** – (str, optional) Subfolder within the repository where the data is stored.
- **proxy_dict** – (dict, optional) Dictionary with keys as either 'http' or 'https' and values as the proxy server. This is useful if you are on a work computer behind a firewall and need to use a proxy out to download data.
- **kws** – (dict, optional) Keywords passed to xarray.open_dataset

Returns The desired xarray dataset.

Examples

```
>>> from climpred.tutorial import load_dataset()
>>> proxy_dict = {'http': '127.0.0.1'}
>>> ds = load_dataset('FOSI-SST', cache=False, proxy_dict=proxy_dict)
```

2.12 What's New

2.12.1 climpred v1.2.1 (2020-01-07)

Depreciated

- `mad` no longer a keyword for the median absolute error metric. Users should now use `median_absolute_error`, which is identical to changes in `xskillscore` version 0.0.10. (GH#283) Riley X. Brady
- `pacc` no longer a keyword for the p value associated with the Pearson product-moment correlation, since it is used by the correlation coefficient. (GH#283) Riley X. Brady

- `msss` no longer a keyword for the Murphy's MSSS, since it is reserved for the standard MSSS. (GH#283) Riley X. Brady

New Features

- Metrics `pearson_r_eff_p_value` and `spearman_r_eff_p_value` account for autocorrelation in computing p values. (GH#283) Riley X. Brady
- Metric `effective_sample_size` computes number of independent samples between two time series being correlated. (GH#283) Riley X. Brady
- Added keywords for metrics: (GH#283) Riley X. Brady
  - `'pval'` for `pearson_r_p_value`
  - `['n_eff', 'eff_n']` for `effective_sample_size`
  - `['p_pval_eff', 'pvalue_eff', 'pval_eff']` for `pearson_r_eff_p_value`
  - `['spvalue', 'spval']` for `spearman_r_p_value`
  - `['s_pval_eff', 'spvalue_eff', 'spval_eff']` for `spearman_r_eff_p_value`
  - `'nev'` for `nmse`

Bug Fixes

- `climpred` now requires `xarray` version 0.14.1 so that the `drop_vars()` keyword used in our package does not throw an error. (GH#276) Riley X. Brady
- Update to `xskillscore` version 0.0.10 to fix errors in weighted metrics with pairwise NaNs. (GH#283) Riley X. Brady

Internals/Minor Fixes

- `doc8` added to `pre-commit` to have consistent formatting on `.rst` files. (GH#283) Riley X. Brady
- Remove `proper` attribute on `Metric` class since it isn't used anywhere. (GH#283) Riley X. Brady
- Add testing for effective p values. (GH#283) Riley X. Brady
- Add testing for whether metric aliases are repeated/overwrite each other. (GH#283) Riley X. Brady
- `ppp` changed to `msess`, but keywords allow for `ppp` and `msss` still. (GH#283) Riley X. Brady

Documentation

- Expansion of metrics documentation with much more detail on how metrics are computed, their keywords, references, min/max/perfect scores, etc. (GH#283) Riley X. Brady
- Update terminology page with more information on metrics terminology. (GH#283) Riley X. Brady

2.12.2 climpred v1.2.0 (2019-12-17)

Depreciated

- Abbreviation `pval` depreciated. Use `p_pval` for `pearson_r_p_value` instead. (GH#264) Aaron Spring.

New Features

- Users can now pass a custom `metric` or `comparison` to compute functions. (GH#268) Aaron Spring.
  - See user-defined-metrics and user-defined-comparisons.
- New deterministic metrics (see metrics). (GH#264) Aaron Spring.
  - Spearman ranked correlation (spearman_r)
  - Spearman ranked correlation p-value (spearman_r_p_value)
  - Mean Absolute Deviation (mad)
  - Mean Absolute Percent Error (mape)
  - Symmetric Mean Absolute Percent Error (smape)
- Users can now apply arbitrary `xarray` methods to *`HindcastEnsemble`* and *`PerfectModelEnsemble`*. (GH#243) Riley X. Brady.
  - See the Prediction Ensemble objects demo page.
- Add "getter" methods to *`HindcastEnsemble`* and *`PerfectModelEnsemble`* to retrieve `xarray` datasets from the objects. (GH#243) Riley X. Brady.

```
>>> hind = climpred.tutorial.load_dataset('CESM-DP-SST')
>>> ref = climpred.tutorial.load_dataset('ERSST')
>>> hindcast = climpred.HindcastEnsemble(hind)
>>> hindcast = hindcast.add_reference(ref, 'ERSST')
>>> print(hindcast)
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 ...
ERSST:
    SST      (time) float32 ...
Uninitialized:
    None
>>> print(hindcast.get_initialized())
<xarray.Dataset>
Dimensions:  (init: 64, lead: 10, member: 10)
Coordinates:
* lead     (lead) int32 1 2 3 4 5 6 7 8 9 10
* member   (member) int32 1 2 3 4 5 6 7 8 9 10
* init     (init) float32 1954.0 1955.0 1956.0 1957.0 ... 2015.0 2016.0␣
↪2017.0
Data variables:
    SST      (init, lead, member) float64 ...
>>> print(hindcast.get_reference('ERSST'))
<xarray.Dataset>
Dimensions:  (time: 61)
Coordinates:
* time     (time) int64 1955 1956 1957 1958 1959 ... 2011 2012 2013 2014␣
↪2015
Data variables:
    SST      (time) float32 ...
```

- `metric_kwargs` can be passed to *`Metric`*. (GH#264) Aaron Spring.
  - See `metric_kwargs` under metrics.

Bug Fixes

- *compute_metric()* doesn't drop coordinates from the initialized hindcast ensemble anymore. (GH#258) Aaron Spring.
- Metric uacc does not crash when ppp negative anymore. (GH#264) Aaron Spring.
- Update xskillscore to version 0.0.9 to fix all-NaN issue with pearson_r and pearson_r_p_value when there's missing data. (GH#269) Riley X. Brady.

Internals/Minor Fixes

- Rewrote *varweighted_mean_period()* based on xrft. Changed time_dim to dim. Function no longer drops coordinates. (GH#258) Aaron Spring
- Add dim='time' in *dpp()*. (GH#258) Aaron Spring
- Comparisons m2m, m2e rewritten to not stack dims into supervector because this is now done in xskillscore. (GH#264) Aaron Spring
- Add tqdm progress bar to *bootstrap_compute()*. (GH#244) Aaron Spring
- Remove inplace behavior for *HindcastEnsemble* and *PerfectModelEnsemble*. (GH#243) Riley X. Brady
  - See demo page on prediction ensemble objects
- Added tests for chunking with dask. (GH#258) Aaron Spring
- Fix test issues with esmpy 8.0 by forcing esmpy 7.1 (GH#269). Riley X. Brady
- Rewrote metrics and comparisons as classes to accomodate custom metrics and comparisons. (GH#268) Aaron Spring
  - See user-defined-metrics and user-defined-comparisons.

Documentation

- Add examples notebook for temporal and spatial smoothing. (GH#244) Aaron Spring
- Add documentation for computing a metric over a specified dimension. (GH#244) Aaron Spring
- Update API to be more organized with individual function/class pages. (GH#243) Riley X. Brady.
- Add page describing the *HindcastEnsemble* and *PerfectModelEnsemble* objects more clearly. (GH#243) Riley X. Brady
- Add page for publications and helpful links. (GH#270) Riley X. Brady.

2.12.3 climpred v1.1.0 (2019-09-23)

Features

- Write information about skill computation to netcdf attributes(GH#213) Aaron Spring
- Temporal and spatial smoothing module (GH#224) Aaron Spring
- Add metrics *brier_score*, *threshold_brier_score* and *crpss_es* (GH#232) Aaron Spring

- Allow *compute_hindcast* and *compute_perfect_model* to specify which dimension *dim* to calculate metric over (GH#232) Aaron Spring

Bug Fixes

- Correct implementation of probabilistic metrics from *xskillscore* in *compute_perfect_model*, *bootstrap_perfect_model*, *compute_hindcast* and *bootstrap_hindcast*, now requires xskillscore>=0.05 (GH#232) Aaron Spring

Internals/Minor Fixes

- Rename .stats.DPP to dpp (GH#232) Aaron Spring
- Add *matplotlib* as a main dependency so that a direct pip installation works (GH#211) Riley X. Brady.
- `climpred` is now installable from conda-forge (GH#212) Riley X. Brady.
- Fix erroneous descriptions of sample datasets (GH#226) Riley X. Brady.
- Benchmarking time and peak memory of compute functions with *asv* (GH#231) Aaron Spring

Documentation

- Add scope of package to docs for clarity for users and developers. (GH#235) Riley X. Brady.

2.12.4 climpred v1.0.1 (2019-07-04)

Bug Fixes

- Accomodate for lead-zero within the `lead` dimension (GH#196) Riley X. Brady.
- Fix issue with adding uninitialized ensemble to `HindcastEnsemble` object (GH#199) Riley X. Brady.
- Allow `max_dof` keyword to be passed to `compute_metric` and `compute_persistence` for `HindcastEnsemble` (GH#199) Riley X. Brady.

Internals/Minor Fixes

- Force `xskillscore` version 0.0.4 or higher to avoid `ImportError` (GH#204) Riley X. Brady.
- Change `max_dfs` keyword to `max_dof` (GH#199) Riley X. Brady.
- Add testing for `HindcastEnsemble` and `PerfectModelEnsemble` (GH#199) Riley X. Brady

2.12.5 climpred v1.0.0 (2019-07-03)

`climpred` v1.0.0 represents the first stable release of the package. It includes `HindcastEnsemble` and `PerfectModelEnsemble` objects to perform analysis with. It offers a suite of deterministic and probabilistic metrics that are optimized to be run on single time series or grids of data (e.g., lat, lon, and depth). Currently, `climpred` only supports annual forecasts.

Features

- Bootstrap prediction skill based on resampling with replacement consistently in `ReferenceEnsemble` and `PerfectModelEnsemble`. (GH#128) Aaron Spring
- Consistent bootstrap function for `climpred.stats` functions via `bootstrap_func` wrapper. (GH#167) Aaron Spring
- many more metrics: `_msss_murphy`, `_less` and probabilistic `_crps`, `_crpss` (GH#128) Aaron Spring

Bug Fixes

- `compute_uninitialized` now trims input data to the same time window. (GH#193) Riley X. Brady
- `rm_poly` now properly interpolates/fills NaNs. (GH#192) Riley X. Brady

Internals/Minor Fixes

- The `climpred` version can be printed. (GH#195) Riley X. Brady
- Constants are made elegant and pushed to a separate module. (GH#184) Andrew Huang
- Checks are consolidated to their own module. (GH#173) Andrew Huang

Documentation

- Documentation built extensively in multiple PRs.

2.12.6 climpred v0.3 (2019-04-27)

`climpred` v0.3 really represents the entire development phase leading up to the version 1 release. This was done in collaboration between Riley X. Brady, Aaron Spring, and Andrew Huang. Future releases will have less additions.

Features

- Introduces object-oriented system to `climpred`, with classes `ReferenceEnsemble` and `PerfectModelEnsemble`. (GH#86) Riley X. Brady
- Expands bootstrapping module for perfect-module configurations. (GH#78, GH#87) Aaron Spring
- Adds functions for computing Relative Entropy (GH#73) Aaron Spring
- Sets more intelligible dimension expectations for `climpred` (GH#98, GH#105) Riley X. Brady and Aaron Spring:
  - `init`: initialization dates for the prediction ensemble
  - `lead`: retrospective forecasts from prediction ensemble; returned dimension for prediction calculations
  - `time`: time dimension for control runs, references, etc.
  - `member`: ensemble member dimension.
- Updates `open_dataset` to display available dataset names when no argument is passed. (GH#123) Riley X. Brady
- Change `ReferenceEnsemble` to `HindcastEnsemble`. (GH#124) Riley X. Brady

- Add probabilistic metrics to `climpred`. (GH#128) Aaron Spring
- Consolidate separate perfect-model and hindcast functions into singular functions (GH#128) Aaron Spring
- Add option to pass proxy through to `open_dataset` for firewalled networks. (GH#138) Riley X. Brady

Bug Fixes

- `xr_rm_poly` can now operate on Datasets and with multiple variables. It also interpolates across NaNs in time series. (GH#94) Andrew Huang
- Travis CI, `treon`, and `pytest` all run for automated testing of new features. (GH#98, GH#105, GH#106) Riley X. Brady and Aaron Spring
- Clean up `check_xarray` decorators and make sure that they work. (GH#142) Andrew Huang
- Ensures that `help()` returns proper docstring even with decorators. (GH#149) Andrew Huang
- Fixes bootstrap so p values are correct. (GH#170) Aaron Spring

Internals/Minor Fixes

- Adds unit testing for all perfect-model comparisons. (GH#107) Aaron Spring
- Updates CESM-LE uninitialized ensemble sample data to have 34 members. (GH#113) Riley X. Brady
- Adds MPI-ESM hindcast, historical, and assimilation sample data. (GH#119) Aaron Spring
- Replaces `check_xarray` with a decorator for checking that input arguments are xarray objects. (GH#120) Andrew Huang
- Add custom exceptions for clearer error reporting. (GH#139) Riley X. Brady
- Remove "xr" prefix from stats module. (GH#144) Riley X. Brady
- Add codecoverage for testing. (GH#152) Riley X. Brady
- Update exception messages for more pretty error reporting. (GH#156) Andrew Huang
- Add `pre-commit` and `flake8`/`black` check in CI. (GH#163) Riley X. Brady
- Change `loadutils` module to `tutorial` and `open_dataset` to `load_dataset`. (GH#164) Riley X. Brady
- Remove predictability horizon function to revisit for v2. (GH#165) Riley X. Brady
- Increase code coverage through more testing. (GH#167) Aaron Spring
- Consolidates checks and constants into modules. (GH#173) Andrew Huang

2.12.7 climpred v0.2 (2019-01-11)

Name changed to `climpred`, developed enough for basic decadal prediction tasks on a perfect-model ensemble and reference-based ensemble.

2.12.8 climpred v0.1 (2018-12-20)

Collaboration between Riley Brady and Aaron Spring begins.

2.13 Helpful Links

We hope to curate in the `climpred` documentation a comprehensive report of terminology, best practices, analysis methods, etc. in the prediction community. Here we suggest other resources for initialized prediction of the Earth system to round out the information provided in our documentation.

2.13.1 Forecast Verification

- CAWCR Forecast Verification Overview: A nice overview of forecast verification, including a suite of metrics and their derivation.

2.14 Publications Using `climpred`

Below is a list of publications that have made use of `climpred` in their analysis. You can nod to `climpred`, e.g., in your acknowledgements section to help build the community. The main developers of the package intend to release a manuscript documenting `climpred` in 2020 with a citable DOI, so this can be referenced in the future.

Feel free to open a Pull Request to add your publication to the list!

2.14.1 2019

- Brady, R. X., Lovenduski, N. S., Yeager, S. G., Long, M. C., & Lindsay, K. (2019, October 10). Skillful multiyear predictions of ocean acidification in the California Current System. https://doi.org/10.31223/osf.io/3m2h7

2.15 Contribution Guide

Contributions are highly welcomed and appreciated. Every little help counts, so do not hesitate! You can make a high impact on `climpred` just by using it and reporting issues.

The following sections cover some general guidelines regarding development in `climpred` for maintainers and contributors. Nothing here is set in stone and can't be changed. Feel free to suggest improvements or changes in the workflow.

Contribution links

- *Contribution Guide*
  - *Feature requests and feedback*
  - *Report bugs*
  - *Fix bugs*
  - *Write documentation*
  - *Preparing Pull Requests*

2.15.1 Feature requests and feedback

We are eager to hear about your requests for new features and any suggestions about the API, infrastructure, and so on. Feel free to submit these as issues with the label "feature request."

Please make sure to explain in detail how the feature should work and keep the scope as narrow as possible. This will make it easier to implement in small PRs.

2.15.2 Report bugs

Report bugs for `climpred` in the issue tracker with the label "bug".

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting, specifically the Python interpreter version, installed libraries, and `climpred` version.
- Detailed steps to reproduce the bug.

If you can write a demonstration test that currently fails but should passm that is a very useful commit to make as well, even if you cannot fix the bug itself.

2.15.3 Fix bugs

Look through the GitHub issues for bugs.

Talk to developers to find out how you can fix specific bugs.

2.15.4 Write documentation

`climpred` could always use more documentation. What exactly is needed?

- More complementary documentation. Have you perhaps found something unclear?
- Docstrings. There can never be too many of them.
- Example notebooks with different Earth System Models, lead times, etc. – they're all very appreciated.

You can also edit documentation files directly in the GitHub web interface, without using a local copy. This can be convenient for small fixes.

Our documentation is written in reStructuredText. You can follow our conventions in already written documents. Some helpful guides are located here and here.

Note: Build the documentation locally with the following command:

```
$ conda env update -f ci/environment-dev-3.6.yml
$ cd docs
$ make html
```

The built documentation should be available in the `docs/build/`.

If you need to add new functions to the API, run `sphinx-autogen -o api api.rst` from the `docs/source` directory and add the functions to `api.rst`.

2.15.5 Preparing Pull Requests

1. Fork the climpred GitHub repository. It's fine to use `climpred` as your fork repository name because it will live under your user.
2. Clone your fork locally using git, connect your repository to the upstream (main project), and create a branch:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/climpred.git
$ cd climpred
$ git remote add upstream git@github.com:bradyrx/climpred.git

# now, to fix a bug or add feature create your own branch off "master":

$ git checkout -b your-bugfix-feature-branch-name master
```

   If you need some help with Git, follow this quick start guide: https://git.wiki.kernel.org/index.php/QuickStart

3. Install dependencies into a new conda environment:

```
$ conda env update -f ci/environment-dev-3.7.yml
$ conda activate climpred-dev
```

4. Make an editable install of climpred by running:

```
$ pip install -e .
```

5. Install pre-commit and its hook on the `climpred` repo:

```
$ pip install --user pre-commit
$ pre-commit install
```

   Afterwards `pre-commit` will run whenever you commit.

   https://pre-commit.com/ is a framework for managing and maintaining multi-language pre-commit hooks to ensure code-style and code formatting is consistent.

   > Now you have an environment called `climpred-dev` that you can work in. You'll need to make sure to activate that environment next time you want to use it after closing the terminal or your system.
   >
   > You can now edit your local working copy and run/add tests as necessary. Please follow PEP-8 for naming. When committing, `pre-commit` will modify the files as needed, or will generally be quite clear about what you need to do to pass the commit test.

6. Break your edits up into reasonably sized commits.

   > $ git commit -a -m "<commit message>" $ git push -u

7. Run all the tests

   Now running tests is as simple as issuing this command:

```
$ coverage run --source climpred -m py.test
```

   This command will run tests via the "pytest" tool against Python 3.6.

8. Create a new changelog entry in `CHANGELOG.rst`:

   - The entry should be entered as:

<description> (:pr:`#<pull request number>`) `<author's names>`_

where <description> is the description of the PR related to the change and <pull request number> is the pull request number and <author's names> are your first and last names.

- Add yourself to list of authors at the end of CHANGELOG.rst file if not there yet, in alphabetical order.

1. Add yourself to the *contributors <https://climpred.readthedocs.io/en/latest/contributors.html>*_ list via docs/source/contributors.rst.

1. Finally, submit a pull request through the GitHub website using this data:

```
head-fork: YOUR_GITHUB_USERNAME/climpred
compare: your-branch-name

base-fork: bradyrx/climpred
base: master
```

Note that you can create the Pull Request while you're working on this. The PR will update as you add more commits. climpred developers and contributors can then review your code and offer suggestions.

2.16 Release Procedure

We follow semantic versioning, e.g., v1.0.0. A major version causes incompatible API changes, a minor version adds functionality, and a patch covers bug fixes.

1. Create a new branch release-vX.x.x with the version for the release.

- Update *CHANGELOG.rst*
- Make sure all new changes, features are reflected in the documentation.

1. Open a new pull request for this branch targeting *master*
2. After all tests pass and the PR has been approved, merge the PR into master
3. Tag a release and push to github:

```
$ git tag -a v1.0.0 -m "Version 1.0.0"
$ git push origin master --tags
```

4. Build and publish release on PyPI:

```
$ git clean -xfd  # remove any files not checked into git
$ python setup.py sdist bdist_wheel --universal  # build package
$ twine upload dist/*  # register and push to pypi
```

5. Update the stable branch (used by ReadTheDocs):

```
$ git checkout stable
$ git rebase master
$ git push -f origin stable
$ git checkout master
```

6. Update climpred conda-forge feedstock

- Fork climpred-feedstock repository
- Clone this fork and edit recipe:

```
$ git clone git@github.com:username/climpred-feedstock.git
$ cd climpred-feedstock
$ cd recipe
$ # edit meta.yaml
```

- Update version
- Get sha256 from pypi.org for climpred
- Check that `requirements.txt` from the main `climpred` repo is accounted for in `meta.yaml` from the feedstock.
- Fill in the rest of information as described here
- Commit and submit a PR

2.17 Contributors

2.17.1 Core Developers

- Riley X. Brady (github)
- Aaron Spring (github)

2.17.2 Contributors

- Andrew Huang (github)

For a list of all the contributions, see the github contribution graph.

Bibliography

[Jolliffe2011] Ian T. Jolliffe and David B. Stephenson. Forecast Verification: A Practitioner's Guide in Atmospheric Science. John Wiley & Sons, Ltd, Chichester, UK, December 2011. ISBN 978-1-119-96000-3 978-0-470-66071-3. URL: http://doi.wiley.com/10.1002/9781119960003.

[Murphy1988] Allan H. Murphy. Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient. Monthly Weather Review, 116(12):2417–2424, December 1988. https://doi.org/10/fc7mxd.

[Boer2016] Boer, G. J., D. M. Smith, C. Cassou, F. Doblas-Reyes, G. Danabasoglu, B. Kirtman, Y. Kushnir, et al. "The Decadal Climate Prediction Project (DCPP) Contribution to CMIP6." Geosci. Model Dev. 9, no. 10 (October 25, 2016): 3751–77. https://doi.org/10/f89qdf.

[Bushuk2018] Mitchell Bushuk, Rym Msadek, Michael Winton, Gabriel Vecchi, Xiaosong Yang, Anthony Rosati, and Rich Gudgel. Regional Arctic sea–ice prediction: potential versus operational seasonal forecast skill. Climate Dynamics, June 2018. https://doi.org/10/gd7hfq.

[Griffies1997] S. M. Griffies and K. Bryan. A predictability study of simulated North Atlantic multidecadal variability. Climate Dynamics, 13(7-8):459–487, August 1997. https://doi.org/10/ch4kc4.

[Seferian2018] Roland Séférian, Sarah Berthet, and Matthieu Chevallier. Assessing the Decadal Predictability of Land and Ocean Carbon Uptake. Geophysical Research Letters, March 2018. https://doi.org/10/gdb424.

[Griffies1997] Griffies, S. M., and K. Bryan. "A Predictability Study of Simulated North Atlantic Multidecadal Variability." Climate Dynamics 13, no. 7–8 (August 1, 1997): 459–87. https://doi.org/10/ch4kc4

[Boer2016] Boer, G. J., Smith, D. M., Cassou, C., Doblas-Reyes, F., Danabasoglu, G., Kirtman, B., Kushnir, Y., Kimoto, M., Meehl, G. A., Msadek, R., Mueller, W. A., Taylor, K. E., Zwiers, F., Rixen, M., Ruprich-Robert, Y., and Eade, R.: The Decadal Climate Prediction Project (DCPP) contribution to CMIP6, Geosci. Model Dev., 9, 3751-3777, https://doi.org/10.5194/gmd-9-3751-2016, 2016.

[Jolliffe2011] Ian T. Jolliffe and David B. Stephenson. Forecast Verification: A Practitioner's Guide in Atmospheric Science. John Wiley & Sons, Ltd, Chichester, UK, December 2011. ISBN 978-1-119-96000-3 978-0-470-66071-3. URL: http://doi.wiley.com/10.1002/9781119960003.

[Meehl2013] Meehl, G. A., Goddard, L., Boer, G., Burgman, R., Branstator, G., Cassou, C., … & Karspeck, A. (2014). Decadal climate prediction: an update from the trenches. Bulletin of the American Meteorological Society, 95(2), 243-267. https://doi.org/10.1175/BAMS-D-12-00241.1.

[Murphy1985] Murphy, Allan H., and Daan, H. "Forecast evaluation." Probability, Statistics, and Decision Making in the Atmospheric Sciences, A. H. Murphy and R. W. Katz, Eds., Westview Press, 379-437. https://doi.org.

[Murphy1988] Murphy, Allan H. "Skill Scores Based on the Mean Square Error and Their Relationships to the Correlation Coefficient." Monthly Weather Review 116, no. 12 (December 1, 1988): 2417–24. https://doi.org/10/fc7mxd.

[Jolliffe2012] Jolliffe, Ian T., and David B. Stephenson, eds. Forecast verification: a practitioner's guide in atmospheric science. John Wiley & Sons, 2012.

[Yuan2016] Yuan, Xiaojun, et al. "Arctic sea ice seasonal prediction by a linear Markov model." Journal of Climate 29.22 (2016): 8151-8173.

Index

Symbols

__init__() (*climpred.classes.HindcastEnsemble method*), 68, 69
__init__() (*climpred.classes.PerfectModelEnsemble method*), 72, 73
__init__() (*climpred.comparisons.Comparison method*), 85
__init__() (*climpred.metrics.Metric method*), 83
_bias_slope() (*in module climpred.metrics*), 55
_brier_score() (*in module climpred.metrics*), 60
_conditional_bias() (*in module climpred.metrics*), 54
_crps() (*in module climpred.metrics*), 57
_crpss() (*in module climpred.metrics*), 58
_crpss_es() (*in module climpred.metrics*), 59
_effective_sample_size() (*in module climpred.metrics*), 38
_get_norm_factor() (*in module climpred.metrics*), 84
_mae() (*in module climpred.metrics*), 44
_mape() (*in module climpred.metrics*), 50
_median_absolute_error() (*in module climpred.metrics*), 45
_mse() (*in module climpred.metrics*), 43
_msess() (*in module climpred.metrics*), 49
_msess_murphy() (*in module climpred.metrics*), 56
_nmae() (*in module climpred.metrics*), 47
_nmse() (*in module climpred.metrics*), 46
_nrmse() (*in module climpred.metrics*), 48
_pearson_r() (*in module climpred.metrics*), 36
_pearson_r_eff_p_value() (*in module climpred.metrics*), 39
_pearson_r_p_value() (*in module climpred.metrics*), 37
_rmse() (*in module climpred.metrics*), 43
_smape() (*in module climpred.metrics*), 51
_spearman_r() (*in module climpred.metrics*), 40
_spearman_r_eff_p_value() (*in module climpred.metrics*), 41
_spearman_r_p_value() (*in module climpred.metrics*), 41
_std_ratio() (*in module climpred.metrics*), 53
_threshold_brier_score() (*in module climpred.metrics*), 61
_uacc() (*in module climpred.metrics*), 52
_unconditional_bias() (*in module climpred.metrics*), 54

A

add_control() (*climpred.classes.PerfectModelEnsemble method*), 73
add_reference() (*climpred.classes.HindcastEnsemble method*), 69
add_uninitialized() (*climpred.classes.HindcastEnsemble method*), 69
autocorr() (*in module climpred.stats*), 86

B

bootstrap() (*climpred.classes.PerfectModelEnsemble method*), 74
bootstrap_compute() (*in module climpred.bootstrap*), 76
bootstrap_hindcast() (*in module climpred.bootstrap*), 77
bootstrap_perfect_model() (*in module climpred.bootstrap*), 78
bootstrap_uninit_pm_ensemble_from_control() (*in module climpred.bootstrap*), 79
bootstrap_uninitialized_ensemble() (*in module climpred.bootstrap*), 80

C

Comparison (*class in climpred.comparisons*), 85
compute_hindcast() (*in module climpred.prediction*), 81
compute_metric() (*climpred.classes.HindcastEnsemble method*), 70

compute_metric() (*climpred.classes.PerfectModelEnsemble method*), 74
compute_perfect_model() (*in module climpred.prediction*), 81
compute_persistence() (*climpred.classes.HindcastEnsemble method*), 70
compute_persistence() (*climpred.classes.PerfectModelEnsemble method*), 75
compute_persistence() (*in module climpred.prediction*), 82
compute_uninitialized() (*climpred.classes.HindcastEnsemble method*), 71
compute_uninitialized() (*climpred.classes.PerfectModelEnsemble method*), 75
compute_uninitialized() (*in module climpred.prediction*), 83
control (*climpred.classes.PerfectModelEnsemble attribute*), 72, 73
corr() (*in module climpred.stats*), 86

D

decorrelation_time() (*in module climpred.stats*), 87
dpp() (*in module climpred.stats*), 87
dpp_threshold() (*in module climpred.bootstrap*), 80

G

generate_uninitialized() (*climpred.classes.PerfectModelEnsemble method*), 75
get_control() (*climpred.classes.PerfectModelEnsemble method*), 73
get_initialized() (*climpred.classes.HindcastEnsemble method*), 69
get_initialized() (*climpred.classes.PerfectModelEnsemble method*), 73
get_reference() (*climpred.classes.HindcastEnsemble method*), 69
get_uninitialized() (*climpred.classes.HindcastEnsemble method*), 70
get_uninitialized() (*climpred.classes.PerfectModelEnsemble method*), 73

H

HindcastEnsemble (*class in climpred.classes*), 68

L

load_dataset() (*in module climpred.tutorial*), 89

M

Metric (*class in climpred.metrics*), 83

P

PerfectModelEnsemble (*class in climpred.classes*), 72

R

reference (*climpred.classes.HindcastEnsemble attribute*), 68, 69
rm_poly() (*in module climpred.stats*), 88
rm_trend() (*in module climpred.stats*), 88

S

smooth() (*climpred.classes.HindcastEnsemble method*), 71
smooth_kws (*climpred.classes.HindcastEnsemble attribute*), 71

U

uninitialized (*climpred.classes.HindcastEnsemble attribute*), 68
uninitialized (*climpred.classes.PerfectModelEnsemble attribute*), 72
uninitialized (*in module climpred.classes*), 69, 73

V

varweighted_mean_period() (*in module climpred.stats*), 88
varweighted_mean_period_threshold() (*in module climpred.bootstrap*), 80