climpred

Sep 23, 2019

Getting Started

1 Version 1 Release **3**

2 Installation **5**

Bibliography **53**

Index **55**

CHAPTER 1

Version 1 Release

v1.0.1 of `climpred` is our first bare-bones release to the community. **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. 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()

# detrend
obs = climpred.stats.rm_trend(obs, dim='time')
hind = climpred.stats.rm_trend(hind, dim='init')
```

We can now create a `HindcastEnsemble` object and add our references.

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

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, member) float64 0.005165 0.03014 ... 0.1842 0.1812
observations:
    SST      (time) float32 -0.061960407 -0.023283795 ... 0.072058104 0.165859
Uninitialized:
    None
```

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

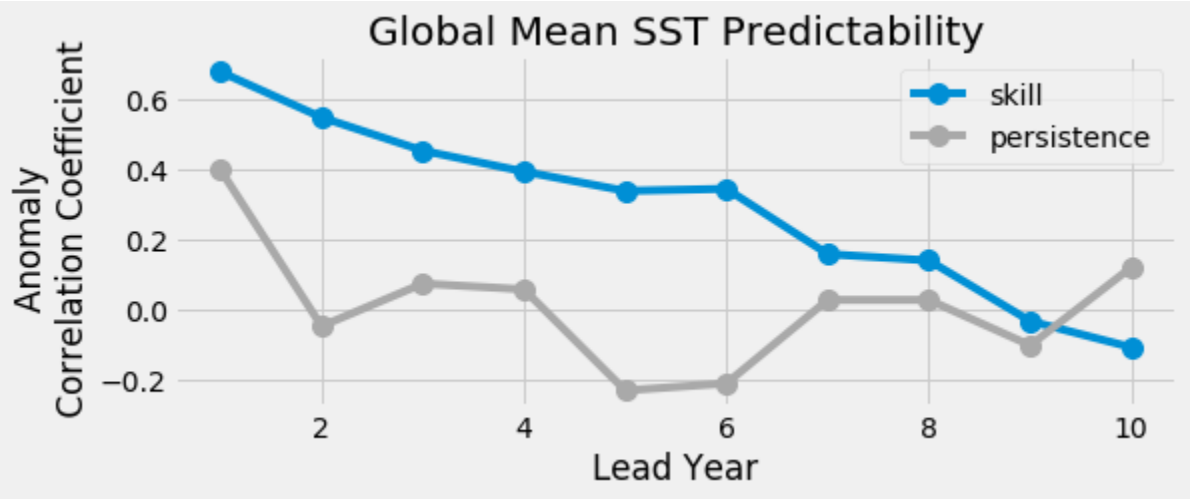
[7]:
```
init = hindcast.compute_metric(metric='pr')
persistence = hindcast.compute_persistence(metric='pr')
```

[8]:
```
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',
```

(continues on next page)

(continued from previous page)

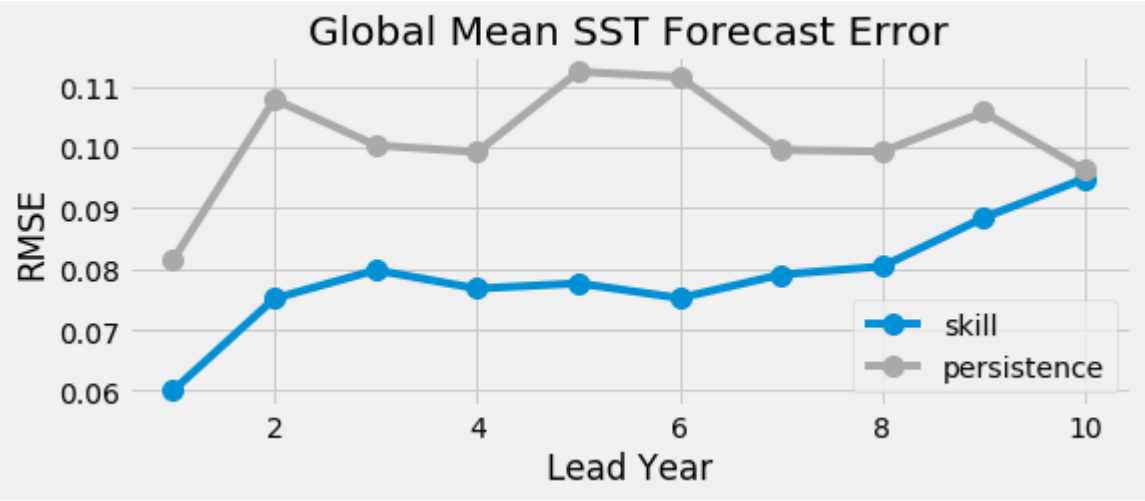
```
       ylabel='Anomaly \n Correlation Coefficient',
       xlabel='Lead Year')
plt.show()
```



We can also check error in our forecasts.

[9]:
```
init = hindcast.compute_metric(metric='rmse')
persistence = hindcast.compute_persistence(metric='rmse')
```

[10]:
```
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 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')
```

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

[4]:
```
varname = 'tos'
area = 'North_Atlantic'
period = 'ym'
```

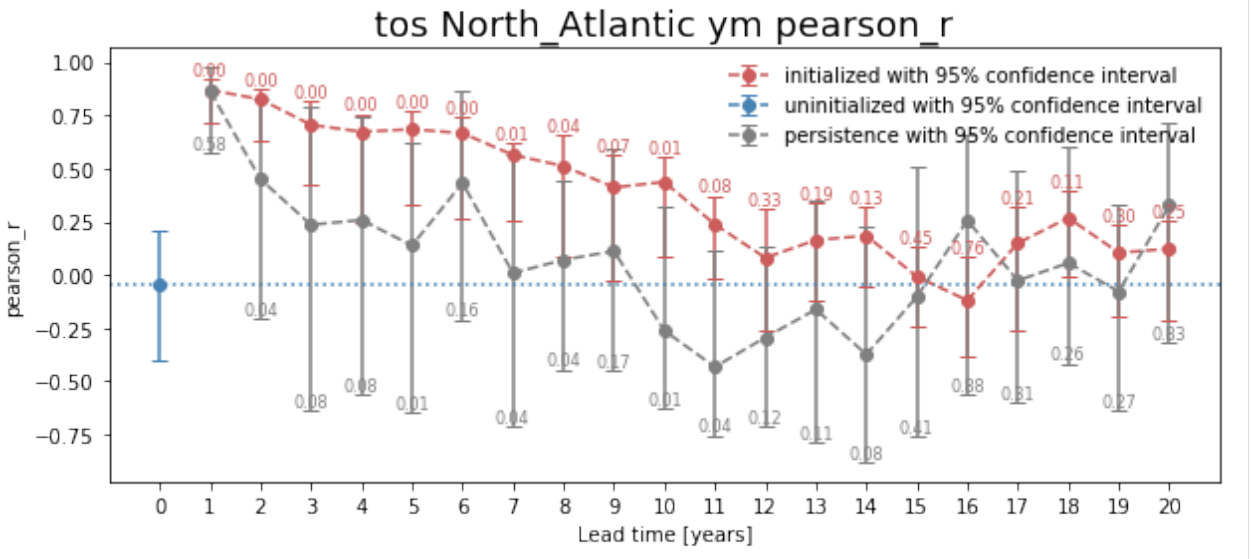
[5]:
```
ds = ds.sel(area=area, period=period)[varname]
control = control.sel(area=area, period=period)[varname]
ds = ds.reset_coords(drop=True)
control = control.reset_coords(drop=True)
```

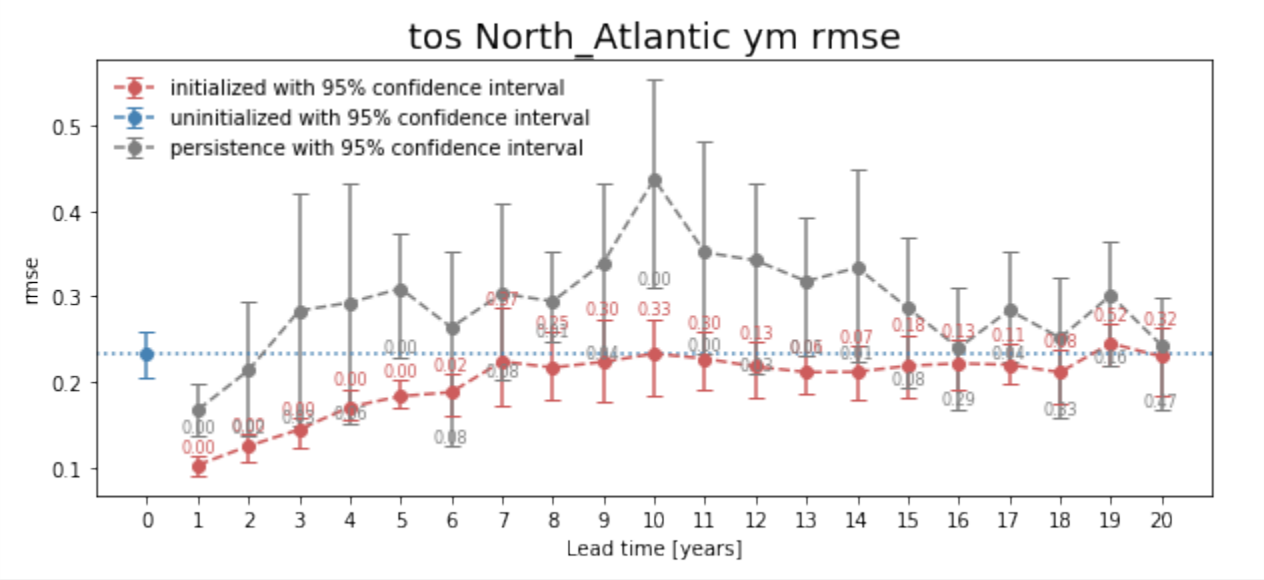
Bootstrapping with Replacement

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.

[7]:
```
for metric in ['pearson_r', 'rmse']:
    bootstrapped = climpred.bootstrap.bootstrap_perfect_model(ds,
                                                              control,
                                                              metric=metric,
                                                              comparison='m2e',
                                                              bootstrap=100,
                                                              sig=95)

    climpred.graphics.plot_bootstrapped_skill_over_leadyear(bootstrapped, sig)
    plt.title(' '.join([varname, area, period, metric]),fontsize=18)
    plt.ylabel(metric)
    plt.show()
```

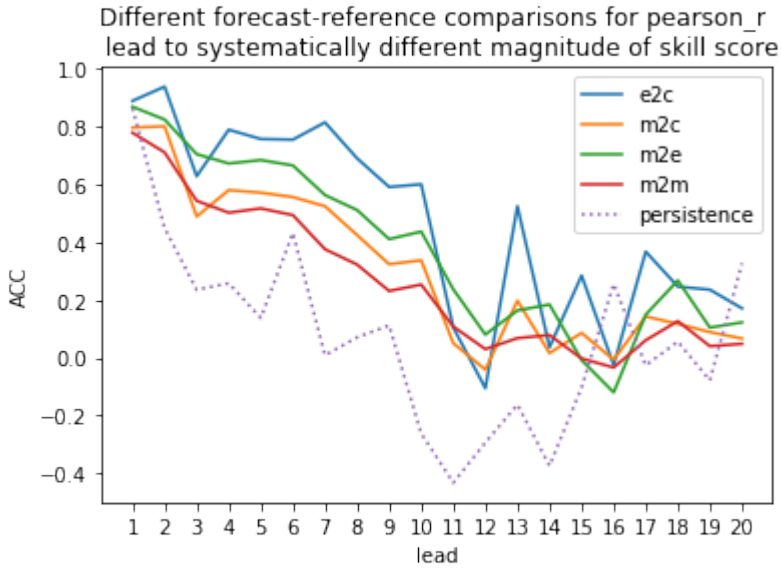




Computing Skill with Different Comparison Methods

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.

[8]:
```
for c in ['e2c','m2c','m2e','m2m']:
    climpred.prediction.compute_perfect_model(ds,
                                              control,
                                              metric='pearson_r',
                                              comparison=c).plot(label=c)
# Persistence computation for a baseline.
climpred.prediction.compute_persistence(ds, control).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).

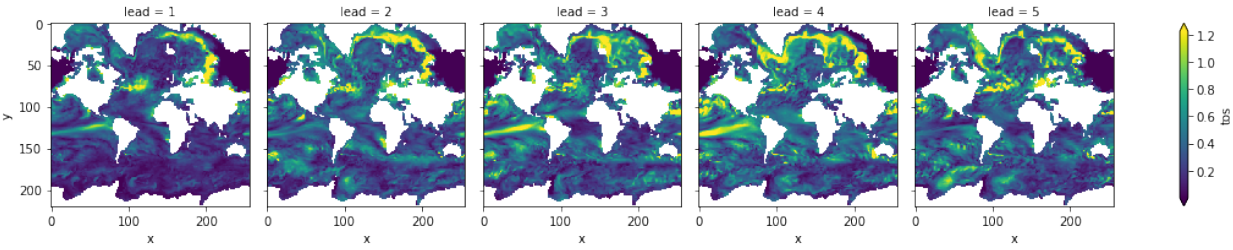
Note: These are very small subsets of the actual MPI simulations so that we could host the sample output maps on Github.

[7]:
```
# Sea surface temperature
ds3d = climpred.tutorial.load_dataset('MPI-PM-DP-3D') \
                        .sel(init=3014) \
                        .expand_dims('init')[varname]
control3d = climpred.tutorial.load_dataset('MPI-control-3D')[varname]
```

Maps of Skill by Lead Year

[10]:
```
climpred.prediction.compute_perfect_model(ds3d,
                                          control3d,
                                          metric='rmse',
                                          comparison='m2e') \
                    .plot(col='lead', robust=True, yincrease=False)
```

[10]:
```
<xarray.plot.facetgrid.FacetGrid at 0x1c2aebe8d0>
```



Slow components of internal variability that indicate potential predictability

Here, we showcase a set of methods to show regions indicating probabilities for decadal predictability.

Diagnostic Potential Predictability (DPP)

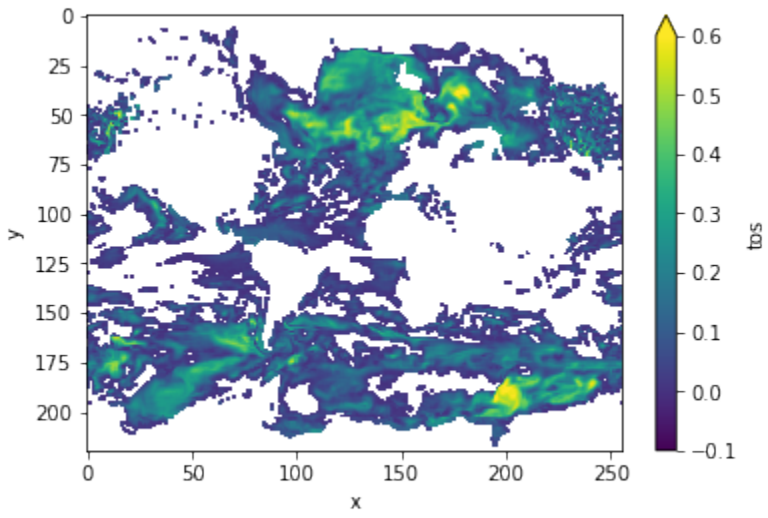
We can first use the [Resplandy 2015] and [Seferian 2018] method for computing DPP, by not "chunking".

[ ]:
```
threshold = climpred.bootstrap.dpp_threshold(control3d,
                                             m=10,
                                             chunk=False,
                                             bootstrap=10)
DPP10 = climpred.stats.dpp(control3d, m=10, chunk=False)
DPP10.where(DPP10 > threshold).plot(yincrease=False, vmin=-0.1, vmax=0.6, cmap=
↪'viridis')
```

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

[19]:
```
threshold = climpred.bootstrap.dpp_threshold(control3d,
                                             m=10,
                                             chunk=True,
                                             bootstrap=bootstrap)
DPP10 = climpred.stats.dpp(control3d, m=10, chunk=True)
DPP10.where(DPP10>0).plot(yincrease=False, vmin=-0.1, vmax=0.6, cmap='viridis')
```

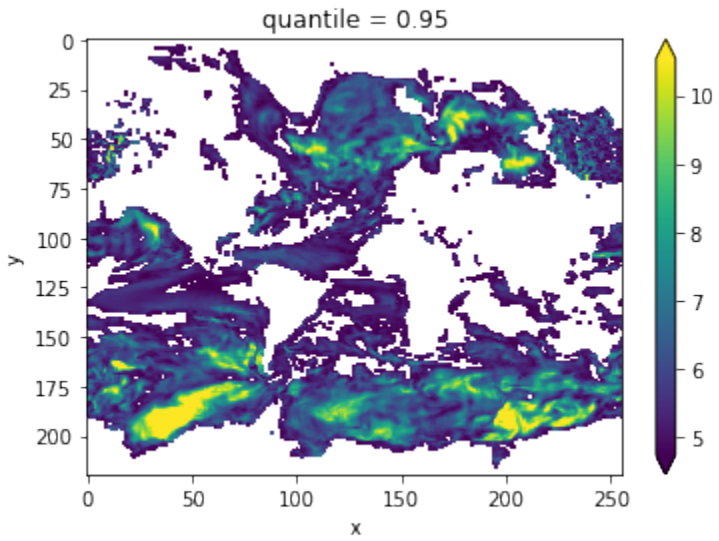
[19]:
```
<matplotlib.collections.QuadMesh at 0x123e8c208>
```



Variance-Weighted Mean Period

[20]:
```
threshold = climpred.bootstrap.varweighted_mean_period_threshold(control3d,
                                                                 bootstrap=bootstrap)
vwmp = climpred.stats.varweighted_mean_period(control3d, time_dim='time')
vwmp.where(vwmp > threshold).plot(yincrease=False, robust=True)
```

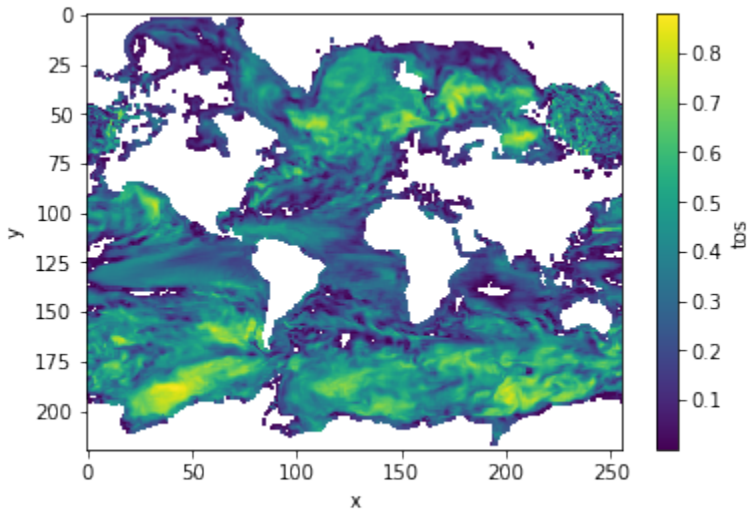
[20]:
```
<matplotlib.collections.QuadMesh at 0x123ea8860>
```



Lag-1 Autocorrelation

[21]:
```
corr_ef = climpred.stats.autocorr(control3d, dim='time')
corr_ef.where(corr_ef>0).plot(yincrease=False, robust=False)
```

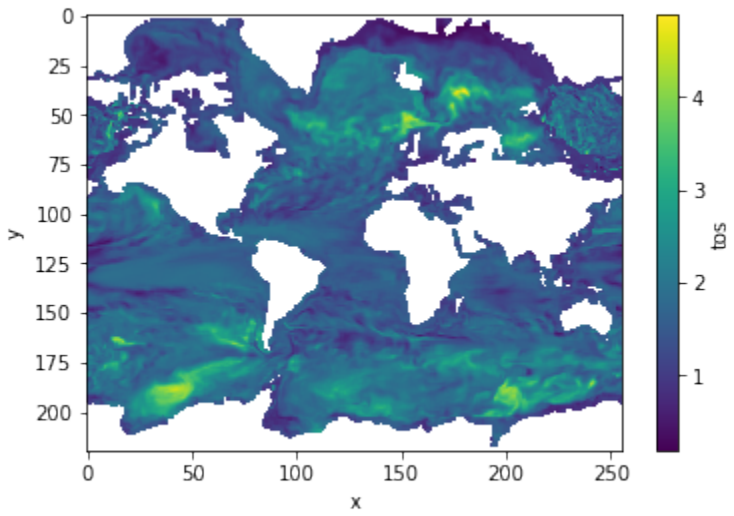
[21]:
```
<matplotlib.collections.QuadMesh at 0x1c314b6860>
```



Decorrelation time

[22]:
```
decorr_time = climpred.stats.decorrelation_time(control3d)
decorr_time.where(decorr_time>0).plot(yincrease=False, robust=False)
```

[22]:
```
<matplotlib.collections.QuadMesh at 0x1c2e3a14a8>
```



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. 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.
3. 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.
4. Goddard, Lisa, et al. "A verification framework for interannual-to-decadal predictions experiments." Climate Dynamics 40.1-2 (2013): 245-272.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.

10. 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.

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
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': decadal prediction ensemble area averages of SST/SSS/AMO.
'MPI-control-3D': decadal prediction ensemble lat/lon/time of SST/SSS/AMO.
'MPI-PM-DP-1D': area averages for the control run of SST/SSS.
'MPI-PM-DP-3D': lat/lon/time for the control run of SST/SSS.
'CESM-DP-SST': decadal prediction ensemble of global mean SSTs.
'CESM-DP-SSS': decadal prediction ensemble of global mean SSS.
'CESM-DP-SST-3D': decadal prediction ensemble of eastern Pacific SSTs.
'CESM-LE': uninitialized ensemble of global mean SSTs.
'MPIESM_miklip_baseline1-hind-SST-global': 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
```

(continues on next page)

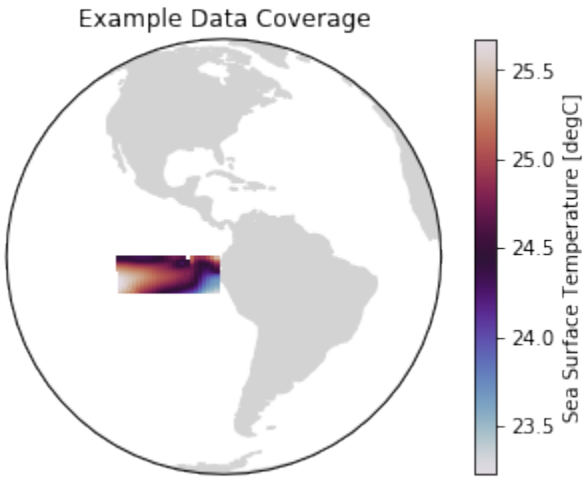
(continued from previous page)

```
    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')

# Remove trend to look at anomalies.
recon = climpred.stats.rm_trend(recon, dim='time')
hind = climpred.stats.rm_trend(hind, dim='init')
```

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.add_reference(recon, 'reconstruction')
print(hindcast)
```

```
<climpred.HindcastEnsemble>
Initialized Ensemble:
    SST      (init, lead, nlat, nlon) float32 -0.29811984 ... 0.5265896
reconstruction:
    SST      (time, nlat, nlon) float32 0.2235269 0.22273289 ... 1.3010706
Uninitialized:
    None
```

Anomaly Correlation Coefficient of SSTs

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

[8]:
```
predictability = hindcast.compute_metric(metric='acc')
# `compute_metric` dropped the TLAT coordinate for some reason. This will
# be fixed in a later version of `climpred`.
predictability['TLAT'] = recon['TLAT']
predictability = predictability.set_coords('TLAT')
```

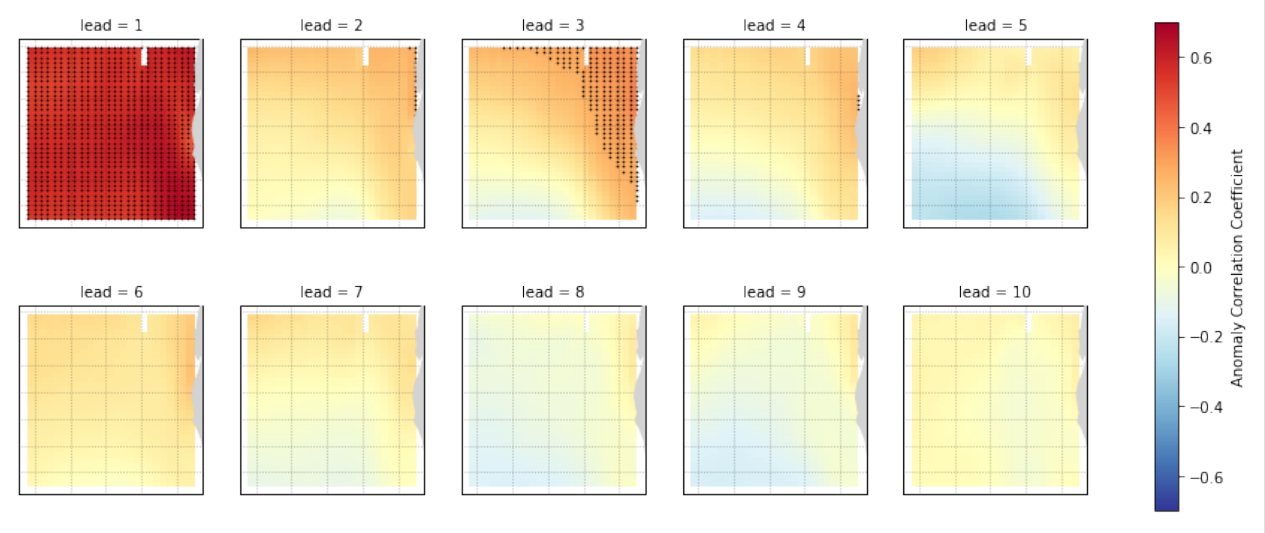
We use the `pval` keyword to get associated p-values for our ACCs. We can then mask our final maps based on $\alpha = 0.05$.

[9]:
```
significance = hindcast.compute_metric(metric='pval')

# `compute_metric` dropped the TLAT coordinate for some reason. This will
# be fixed in a later version of `climpred`.
significance['TLAT'] = recon['TLAT']
significance = significance.set_coords('TLAT')

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

[10]:
```
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())
```

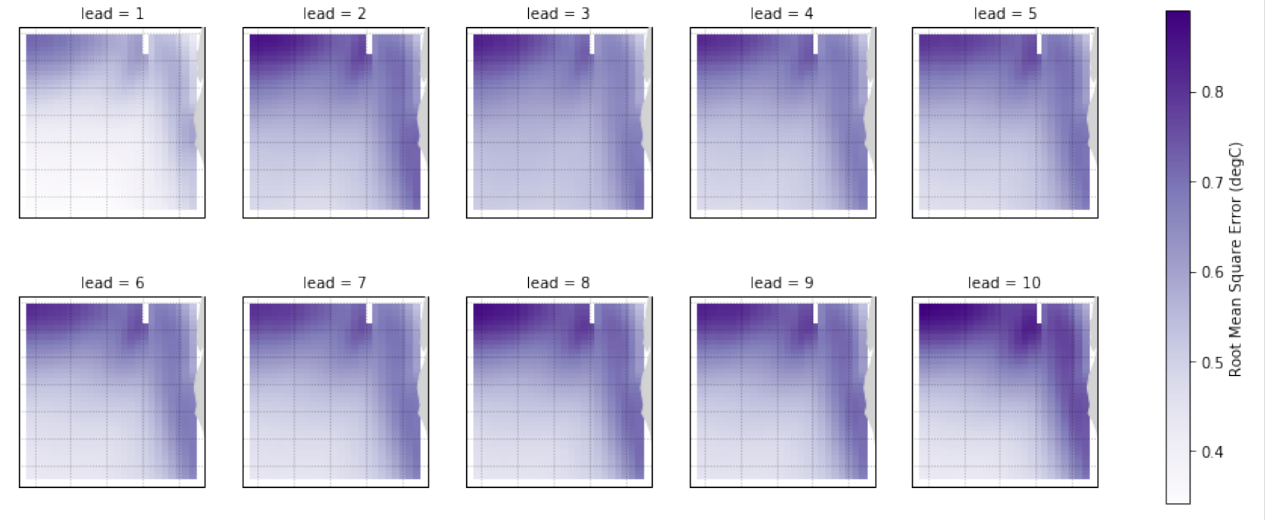


Root Mean Square Error of SSTs

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

[11]:
```
rmse = hindcast.compute_metric(metric='rmse')
rmse['TLAT'] = recon['TLAT']
rmse = rmse.set_coords('TLAT')
```

[12]:
```
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=':')
```



User Guide

- *Setting Up Your Dataset*
- *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 Metrics

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

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 by *`climpred.prediction.compute_hindcast()`* or *`climpred.prediction.compute_perfect_model()`* to specify over which dimensions the `metric` is applied. See *Comparisons*.

2.6.1 Deterministic

Deterministic metrics quantify the level to which the forecast predicts the observations. These metrics are just a special case of probabilistic metrics where a value of 100% is assigned to the forecasted value [Jolliffe2011].

Core Metrics

Anomaly Correlation Coefficient (ACC)

`keyword: 'pearson_r','pr','acc'`

A measure of the linear association between the forecast and observations that is independent of the mean and variance of the individual distributions [Jolliffe2011]. `climpred` uses the Pearson correlation coefficient.

`climpred.metrics.`**`_pearson_r`**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate the Anomaly Correlation Coefficient (ACC).

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

Note: Use metric `pearson_r_p_value` to get the corresponding pvalue.

Range:

- perfect: 1
- min: -1

See also:

- xskillscore.pearson_r
- xskillscore.pearson_r_p_value

Mean Squared Error (MSE)

`keyword: 'mse'`

The average of the squared difference between forecasts and observations. This incorporates both the variance and bias of the estimator.

`climpred.metrics.`**`_mse`**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate the Mean Sqaure Error (MSE).

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

Range:

- perfect: 0
- min: 0
- max: ∞

See also:

- xskillscore.mse

Root Mean Square Error (RMSE)

`keyword: 'rmse'`

The square root of the average of the squared differences between forecasts and observations [Jolliffe2011]. It puts a greater influence on large errors than small errors, which makes this a good choice if large errors are undesirable or one wants to be a more conservative forecaster.

`climpred.metrics._rmse` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Calculate the Root Mean Sqaure Error (RMSE).

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

Range:

- perfect: 0
- min: 0
- max: ∞

See also:

- xskillscore.rmse

Mean Absolute Error (MAE)

`keyword: 'mae'`

The average of the absolute differences between forecasts and observations [Jolliffe2011]. A more robust measure of forecast accuracy than root mean square error or mean square error which is sensitive to large outlier forecast errors [EOS].

`climpred.metrics._mae` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Calculate the Mean Absolute Error (MAE).

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

Range:

- perfect: 0
- min: 0
- max: ∞

See also:

- xskillscore.mae

Derived Metrics

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

Normalized Mean Square Error (NMSE)

`keyword: 'nmse','nev'`

climpred.metrics.**_nmse**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate Normalized MSE (NMSE) = Normalized Ensemble Variance (NEV).

$$NMSE = NEV = \frac{MSE}{\sigma_o^2 \cdot fac}$$

Parameters

- **forecast** (*⋆*) –
- **reference** (*⋆*) –
- **dim** (*⋆*) – dimension to apply metric to
- **comparison** (*⋆*) – name comparison needed for normalization factor (required to be added via **metric_kwargs)

Range:

- 0: perfect forecast: 0
- 0 - 1: better than climatology forecast
- > 1: worse than climatology forecast

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.

Normalized Mean Absolute Error (NMAE)

`keyword: 'nmae'`

climpred.metrics.**_nmae**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Normalized Ensemble Mean Absolute Error metric.

$$NMAE = \frac{MAE}{\sigma_o \cdot fac}$$

Parameters

- **forecast** (*⋆*) –
- **reference** (*⋆*) –
- **dim** (*⋆*) – dimension to apply metric to
- **comparison** (*⋆*) – name comparison needed for normalization factor (required to be added via **metric_kwargs)

Range:

- 0: perfect forecast: 0
- 0 - 1: better than climatology forecast
- > 1: worse than climatology forecast

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.

Normalized Root Mean Square Error (NRMSE)

`keyword: 'nrmse'`

`climpred.metrics._nrmse` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Normalized Root Mean Square Error (NRMSE) metric.

$$NRMSE = \frac{RMSE}{\sigma_o \cdot \sqrt{fac}} = \sqrt{\frac{MSE}{\sigma_o^2 \cdot fac}}$$

Parameters

- **forecast** ($\star$) –
- **reference** ($\star$) –
- **dim** ($\star$) – dimension to apply metric to
- **comparison** ($\star$) – name comparison needed for normalization factor (required to be added via **metric_kwargs)

Range:

- 0: perfect forecast
- 0 - 1: better than climatology forecast
- > 1: worse than climatology forecast

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.

Mean Square Skill Score (MSSS)

`keyword: 'msss','ppp'`

`climpred.metrics._ppp` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Prognostic Potential Predictability (PPP) metric.

$$PPP = 1 - \frac{MSE}{\sigma_{ref}^2 \cdot fac}$$

Parameters

- **forecast** (*) –
- **reference** (*) –
- **dim** (*) – dimension to apply metric to
- **comparison** (*) – name comparison needed for normalization factor (required to be added via **metric_kwargs)

Range:

- 1: perfect forecast
- positive: better than climatology forecast
- negative: worse than climatology forecast

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.
- 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.

Unbiased ACC

`keyword: 'uacc'`

`climpred.metrics._uacc` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate Bushuk's unbiased ACC (uACC).

$$uACC = \sqrt{PPP} = \sqrt{MSSS}$$

Range:

- 1: perfect
- 0 - 1: better than climatology

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.

Murphy decomposition metrics

[Murphy1988] relates the MSSS with ACC and unconditional bias.

Standard Ratio

`keyword: 'std_ratio'`

`climpred.metrics._std_ratio` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Calculate the ratio of standard deviations of reference over forecast.

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

References

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

Unconditional Bias

`keyword: 'bias', 'unconditional_bias', 'u_b'`

`climpred.metrics._bias` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Calculate unconditional bias.

$$bias = f - o$$

Range:

- pos: positive bias
- neg: negative bias
- perfect: 0

References

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

Bias Slope

`keyword: 'bias_slope'`

`climpred.metrics._bias_slope` (*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)
Calculate bias slope between reference and forecast standard deviations.

$$\text{bias slope} = r_{fo} \cdot \text{std ratio}$$

References

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

Conditional Bias

`keyword: 'conditional_bias', c_b'`

climpred.metrics.**_conditional_bias**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate the conditional bias between forecast and reference.

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

References

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

Murphy's Mean Square Skill Score

`keyword: 'msss_murphy'`

climpred.metrics.**_msss_murphy**(*forecast*, *reference*, *dim='svd'*, ***metric_kwargs*)

Calculate Murphy's Mean Square Skill Score (MSSS).

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

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.

2.6.2 Probabilistic

`keyword: 'crps'`

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

Continuous Ranked Probability Score (CRPS) is the probabilistic MSE.

Range:

- perfect: 0
- min: 0
- max: ∞

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.

See also:

- properscoring.crps_ensemble

- xskillscore.crps_ensemble

`keyword: 'crpss'`

`climpred.metrics._crpss` (*forecast*, *reference*, ***metric_kwargs*)
Continuous Ranked Probability Skill Score

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).

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

Parameters

- **forecast** (*) –
- **reference** (*) –
- **gaussian** (*) – Assuming gaussian distribution for baseline skill. Default: True (optional)
- **cdf_or_dist** (*) – distribution to assume if not gaussian. default: scipy.stats.norm
- **xmin, xmax, tol** (*) – only relevant if not gaussian (see xskillscore.crps_quadrature)

Range:

- perfect: 1
- pos: better than climatology forecast
- neg: worse than climatology forecast

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, xmin=-10,
                          xmax=10, tol=1e-6)
```

See also:

- properscoring.crps_ensemble
- xskillscore.crps_ensemble

`keyword: 'crpss_es'`

climpred.metrics.**_crpss_es**(*forecast*, *reference*, ***metric_kwargs*)
CRPSS Ensemble Spread.

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

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

keyword: 'brier_score', 'brier', 'bs'

climpred.metrics.**_brier_score**(*forecast*, *reference*, ***metric_kwargs*)
Calculate Brier score for forecasts on binary reference.

..math: BS(f, o) = (f - o)^2

Parameters

- **forecast** (⋆) –
- **reference** (⋆) –
- **func** (⋆) – function to be applied to reference and forecasts and then mean('member') to get forecasts and reference in interval [0,1]. (required to be added via **metric_kwargs)

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.

Example

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

See also:

- properscoring.brier_score
- xskillscore.brier_score

keyword: 'threshold_brier_score', 'tbs'

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

Calculate the Brier scores of an ensemble for exceeding given thresholds. Provide threshold via metric_kwargs.

$$CRPS(F, x) = \int_z BS(F(z), H(z - x))dz$$

Range:

- perfect: 0
- min: 0
- max: 1

Parameters

- **forecast** (⋆) –
- **reference** (⋆) –
- **threshold** (⋆) – Threshold to check exceedance, see properscoring.threshold_brier_score (required to be added via **metric_kwargs)

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.

Example

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

See also:

- properscoring.threshold_brier_score
- xskillscore.threshold_brier_score

2.6.3 References

2.7 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 *climpred.prediction.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 *climpred.prediction.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.

2.7.1 HindcastEnsemble

_e2r(ds, reference[, stack_dims])	Compare the ensemble mean forecast to a reference in HindcastEnsemble.
_m2r(ds, reference[, stack_dims])	Compares each member individually to a reference in HindcastEnsemble.

2.7.2 PerfectModelEnsemble

_m2e(ds[, supervector_dim, stack_dims])	Create two supervectors to compare all members to ensemble mean while
_m2c(ds[, supervector_dim, control_member, ...])	Create two supervectors to compare all members to control.
_m2m(ds[, supervector_dim, stack_dims])	Create two supervectors to compare all members to all others in turn.
_e2c(ds[, supervector_dim, control_member, ...])	Create two supervectors to compare ensemble mean to control.

2.7.3 References

2.8 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`.

2.8.1 Simulation Design

Initialized Ensemble

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`).

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`).

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.

Reconstruction:

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.

2.8.2 Predictability vs. Prediction skill

(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: This characterizes the current ability of the ensemble forecasting system to predict the real world. This is derived by computing the ACC between the prediction ensemble and observations of the real world [Meehl2013].

2.8.3 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].

2.8.4 References

2.9 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 via `compute_persistence`.

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.

2.9.1 References

Help & Reference

- *API Reference*
- *Contribution Guide*
- *Changelog History*
- *Release Procedure*
- *Contributors*

2.10 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.

2.10.1 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.

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_function()` 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)

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_function()` and (see *Metrics*).
- **comparison** (*str*) – *comparison* name defines what to take as forecast and verification (see `climpred.utils.get_comparison_function()` 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)

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.

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)

2.10.2 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.

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

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

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

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)

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)

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

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

2.10.3 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[, 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(ds[, time_dim])	Calculate the variance weighted mean period of time series.

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.

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).

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

dpp

climpred.stats.**dpp**(*ds*, *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.
- **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.

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.

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.

varweighted_mean_period

climpred.stats.**varweighted_mean_period**(*ds*, *time_dim='time'*)

Calculate the variance weighted mean period of time series.

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

Parameters

- **ds** (*xarray object*) – Time series.
- **time_dim** (*optional str*) – Name of time dimension.

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.

2.10.4 Tutorial

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

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)
```

2.11 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*

2.11.1 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.

2.11.2 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.

2.11.3 Fix bugs

Look through the GitHub issues for bugs.

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

2.11.4 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/`.

2.11.5 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.

2.12 Changelog History

2.12.1 climpred v1.1 (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.

2.12.2 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

2.12.3 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.

2.12.4 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

2.12.5 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.

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

Collaboration between Riley Brady and Aaron Spring begins.

2.13 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 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
    - Fill in the rest of information as described here
    - Commit and submit a PR

2.14 Contributors

2.14.1 Core Developers

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

2.14.2 Contributors

- Andrew Huang (github)

For a list of all the contributions, see the github contribution graph.

Bibliography

[EOS] https://eos.org/opinions/climate-and-other-models-may-be-more-accurate-than-reported

[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.

[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.

[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

_bias() (*in module climpred.metrics*), 26
_bias_slope() (*in module climpred.metrics*), 26
_brier_score() (*in module climpred.metrics*), 29
_conditional_bias() (*in module climpred.metrics*), 27
_crps() (*in module climpred.metrics*), 27
_crpss() (*in module climpred.metrics*), 28
_crpss_es() (*in module climpred.metrics*), 28
_mae() (*in module climpred.metrics*), 22
_mse() (*in module climpred.metrics*), 21
_msss_murphy() (*in module climpred.metrics*), 27
_nmae() (*in module climpred.metrics*), 23
_nmse() (*in module climpred.metrics*), 23
_nrmse() (*in module climpred.metrics*), 24
_pearson_r() (*in module climpred.metrics*), 21
_ppp() (*in module climpred.metrics*), 24
_rmse() (*in module climpred.metrics*), 22
_std_ratio() (*in module climpred.metrics*), 26
_threshold_brier_score() (*in module climpred.metrics*), 29
_uacc() (*in module climpred.metrics*), 25

A

autocorr() (*in module climpred.stats*), 41

B

bootstrap_compute() (*in module climpred.bootstrap*), 36
bootstrap_hindcast() (*in module climpred.bootstrap*), 37
bootstrap_perfect_model() (*in module climpred.bootstrap*), 38
bootstrap_uninit_pm_ensemble_from_control() (*in module climpred.bootstrap*), 39
bootstrap_uninitialized_ensemble() (*in module climpred.bootstrap*), 40

C

compute_hindcast() (*in module climpred.prediction*), 34
compute_perfect_model() (*in module climpred.prediction*), 34
compute_persistence() (*in module climpred.prediction*), 35
compute_uninitialized() (*in module climpred.prediction*), 35
corr() (*in module climpred.stats*), 41

D

decorrelation_time() (*in module climpred.stats*), 42
dpp() (*in module climpred.stats*), 42
dpp_threshold() (*in module climpred.bootstrap*), 40

L

load_dataset() (*in module climpred.tutorial*), 44

R

rm_poly() (*in module climpred.stats*), 43
rm_trend() (*in module climpred.stats*), 43

V

varweighted_mean_period() (*in module climpred.stats*), 43
varweighted_mean_period_threshold() (*in module climpred.bootstrap*), 40