randomforest

March 25, 2024

[ ]:
```
import os
import datetime as dt  # Python standard library datetime  module
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import netCDF4 as nc
import xarray as xr
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import RFE
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from eofs.xarray import Eof
from matplotlib import pyplot as plt
from matplotlib import colors
import matplotlib as mpl
import cartopy
from utils import data_path, normalize_co2, normalize_ch4
from xskillscore import rmse, pearson_r, spearman_r, r2, smape, mae, me, mse
```

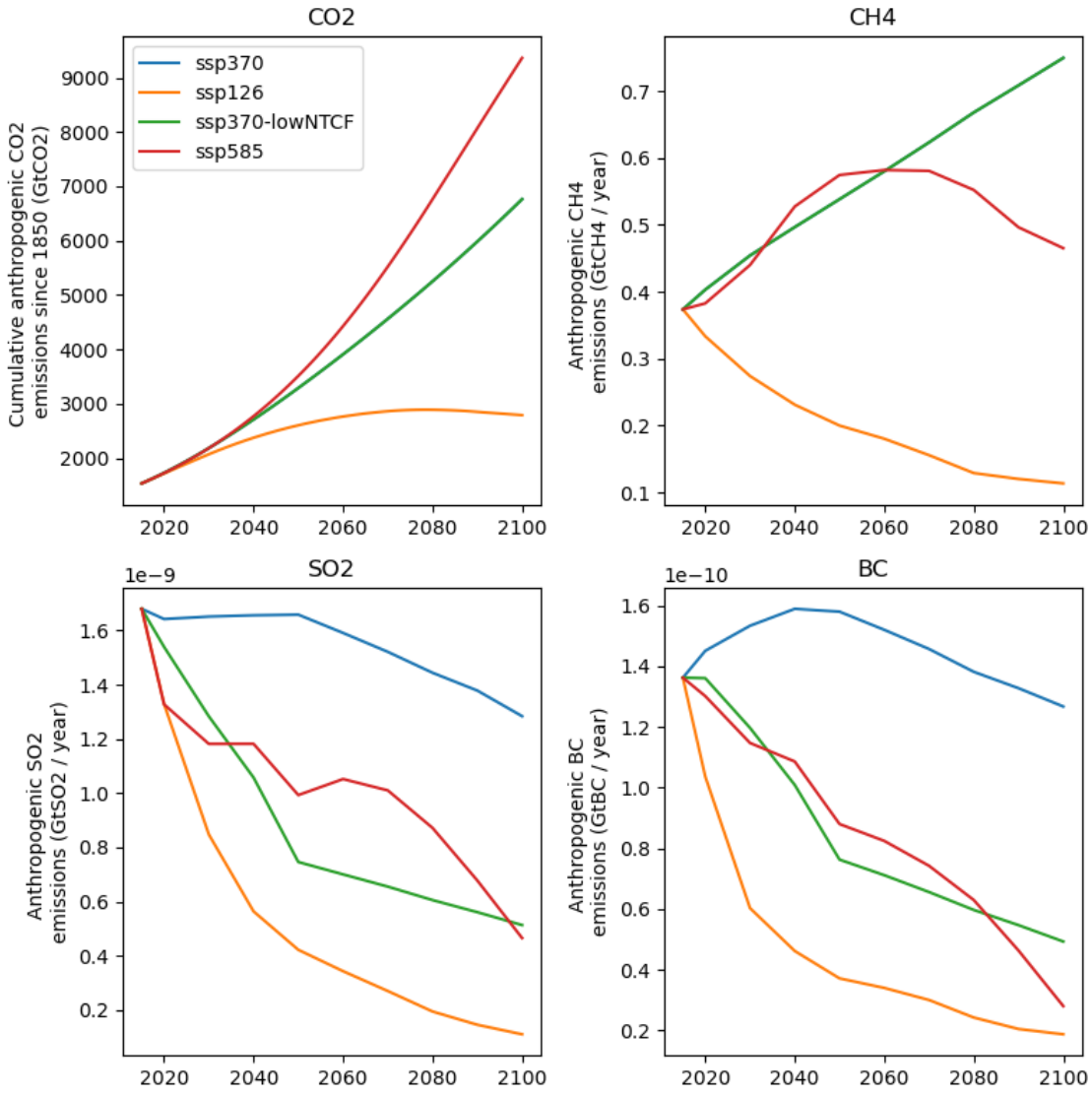
[ ]:
```
def global_mean(ds):
    weights = np.cos(np.deg2rad(ds.lat))
    return ds.weighted(weights).mean(['lat', 'lon'])

def global_sum(ds):
    weights = np.cos(np.deg2rad(ds.lat))
    return ds.weighted(weights).sum(['lat', 'lon'])
```

[ ]:
```
#**parameters & hyperparameters

RSCV = False
path_output='output_path/output.nc'

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 300, num = 5)]
# Number of features to consider at every split
max_features = ['log2', 'sqrt', None]
```

```
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5,55, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [5, 10, 15, 25]
# Minimum number of samples required at each leaf node
min_samples_leaf = [4, 8, 12]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

[ ]:
```
from glob import glob
data_path = '/p/lustre2/shiduan/climatebench/'
inputs = glob(data_path + "inputs_s*.nc")
SECONDS_IN_YEAR = 60*60*24*365 #s

fig, axes = plt.subplots(2, 2, figsize=(8, 8))

for input in inputs:
    label=input.split('_')[1][:-3]
    X = xr.open_dataset(input)
    x = range(2015, 2101)

    weights = np.cos(np.deg2rad(X.latitude))

    axes[0, 0].plot(x, X['CO2'].data, label=label)
    axes[0, 0].set_ylabel("Cumulative anthropogenic CO2 \nemissions since 1850␣
 ↪(GtCO2)")
    axes[0, 1].plot(x, X['CH4'].data, label=label)
    axes[0, 1].set_ylabel("Anthropogenic CH4 \nemissions (GtCH4 / year)")
    # FIXME: Not sure where this factor of 1000 comes from...! Maybe the CEDS␣
 ↪data is really g/m-2/s?
    axes[1, 0].plot(x, X['SO2'].weighted(weights).sum(['latitude',␣
 ↪'longitude']).data*SECONDS_IN_YEAR*1e-9, label=label)
    axes[1, 0].set_ylabel("Anthropogenic SO2 \nemissions (GtSO2 / year)")
    axes[1, 1].plot(x, X['BC'].weighted(weights).sum(['latitude', 'longitude']).
 ↪data*SECONDS_IN_YEAR*1e-9, label=label)
    axes[1, 1].set_ylabel("Anthropogenic BC \nemissions (GtBC / year)")

axes[0, 0].set_title('CO2')
```

```
axes[0, 1].set_title('CH4')
axes[1, 0].set_title('SO2')
axes[1, 1].set_title('BC')
axes[0, 0].legend()
plt.tight_layout()

test_data_path=data_path+'/test/inputs_ssp245.nc'
```



[ ]:
```
# Get one combined historical + ssp585 + ssp126 + ssp370 timeseries for now
```

```
X = xr.concat([xr.open_dataset(data_path + 'inputs_historical.nc'), xr.
 ↪open_dataset(data_path + 'inputs_ssp585.nc'),xr.open_dataset(data_path+␣
 ↪'inputs_ssp126.nc'),xr.open_dataset(data_path+ 'inputs_ssp370.nc'),xr.
 ↪open_dataset(data_path+ 'inputs_hist-aer.nc'),xr.open_dataset(data_path+␣
 ↪'inputs_hist-GHG.nc')], dim='time').compute()

# Take the average member for the historical, ssp585, ssp126, ssp370, hist-aer,␣
 ↪hist-ghg
Y = xr.concat([xr.open_dataset(data_path + 'outputs_historical.nc').
 ↪mean(dim="member"), xr.open_dataset(data_path + 'outputs_ssp585.nc').
 ↪mean(dim="member"),xr.open_dataset(data_path+ 'outputs_ssp126.nc').
 ↪mean(dim="member"),xr.open_dataset(data_path+ 'outputs_ssp370.nc').
 ↪mean(dim="member"),xr.open_dataset(data_path+ 'outputs_hist-aer.nc').
 ↪mean(dim="member"),xr.open_dataset(data_path+ 'outputs_hist-GHG.nc').
 ↪mean(dim="member")], dim='time').compute()


# Convert the precip values to mm/day
Y["pr"] *= 86400
Y["pr90"] *= 86400

X["time"]=np.arange(1, 424 + 165 + 165) ## 165+86+86+86+65+165+1
Y["time"]=np.arange(1, 424 + 165 + 165)
```

[ ]:
```
# Create an EOF solver to do the EOF analysis. Square-root of cosine of
# latitude weights are applied before the computation of EOFs.
bc_solver = Eof(X['BC'])

# Retrieve the leading EOF, expressed as the correlation between the leading
# PC time series and the input SST anomalies at each grid point, and the
# leading PC time series itself.
bc_eofs = bc_solver.eofsAsCorrelation(neofs=5)
bc_pcs = bc_solver.pcs(npcs=5, pcscaling=1)
```

[ ]:
```
# Create an EOF solver to do the EOF analysis. Square-root of cosine of
# latitude weights are applied before the computation of EOFs.
so2_solver = Eof(X['SO2'])

# Retrieve the leading EOF, expressed as the correlation between the leading
# PC time series and the input SST anomalies at each grid point, and the
# leading PC time series itself.
so2_eofs = so2_solver.eofsAsCorrelation(neofs=5)
so2_pcs = so2_solver.pcs(npcs=5, pcscaling=1)
```

[ ]:
```
# Convert the Principle Components of the aerosol emissions (calculated above)␣
 ↪in to Pandas DataFrames
bc_df = bc_pcs.to_dataframe().unstack('mode')
bc_df.columns = [f"BC_{i}" for i in range(5)]

so2_df = so2_pcs.to_dataframe().unstack('mode')
so2_df.columns = [f"SO2_{i}" for i in range(5)]
```

[ ]:
```
# Bring the emissions data back together again and normalise
leading_historical_inputs = pd.DataFrame({
    "CO2": normalize_co2(X["CO2"].data),
    "CH4": normalize_ch4(X["CH4"].data)
}, index=X["CO2"].coords['time'].data)

# Combine with aerosol EOFs
leading_historical_inputs=pd.concat([leading_historical_inputs, bc_df, so2_df],␣
 ↪axis=1)
```

[ ]:
```
def get_rmse(truth, pred):
    weights = np.cos(np.deg2rad(truth.lat))
    return np.sqrt(((truth-pred)**2).weighted(weights).mean(['lat', 'lon'])).
 ↪data.mean()
```

[ ]:
```
y_inp_tas=Y["tas"].stack(dim=["lat", "lon"])
y_inp_pr=Y["pr"].stack(dim=["lat", "lon"])
y_inp_pr90=Y["pr90"].stack(dim=["lat", "lon"])
y_inp_dtr=Y["diurnal_temperature_range"].stack(dim=["lat", "lon"])
```

[ ]:
```
df_y_input_tas = pd.DataFrame(y_inp_tas.to_pandas())
df_y_input_pr = pd.DataFrame(y_inp_pr.to_pandas())
df_y_input_pr90 = pd.DataFrame(y_inp_pr90.to_pandas())
df_y_input_dtr = pd.DataFrame(y_inp_dtr.to_pandas())

Xy_train_tas_ = pd.concat([leading_historical_inputs, df_y_input_tas], axis=1)
Xy_train_pr_ = pd.concat([leading_historical_inputs, df_y_input_pr], axis=1)
Xy_train_pr90_ = pd.concat([leading_historical_inputs, df_y_input_pr90], axis=1)
Xy_train_dtr_ = pd.concat([leading_historical_inputs, df_y_input_dtr], axis=1)
```

[ ]:
```
Xy_train_tas = Xy_train_tas_.to_numpy()
Xy_train_pr = Xy_train_pr_.to_numpy()
Xy_train_pr90 = Xy_train_pr90_.to_numpy()
Xy_train_dtr = Xy_train_dtr_.to_numpy()
```

[ ]:
```
n_inp=leading_historical_inputs.shape[1]
n_iout=Xy_train_tas_.shape[1]

X_train_tas=Xy_train_tas[:,0:n_inp]
```

```
y_train_tas=Xy_train_tas[:,n_inp:n_iout]

X_train_pr=Xy_train_pr[:,0:n_inp]
y_train_pr=Xy_train_pr[:,n_inp:n_iout]

X_train_pr90=Xy_train_pr90[:,0:n_inp]
y_train_pr90=Xy_train_pr90[:,n_inp:n_iout]

X_train_dtr=Xy_train_dtr[:,0:n_inp]
y_train_dtr=Xy_train_dtr[:,n_inp:n_iout]
```

[ ]:
```
np.sum(np.isnan(y_train_pr90))
```

[ ]:
```
0
```

[ ]:
```
# use github parameters
'''reg0 = RandomForestRegressor(random_state=0, bootstrap=True, max_features=1.
 ↪0,
                                **{'n_estimators': 250, 'min_samples_split': 5,␣
 ↪'min_samples_leaf': 7,  'max_depth': 5,})'''
reg0 = RandomForestRegressor(random_state=0, max_features=1.0, bootstrap=True,␣
 ↪n_estimators=250,
                                min_samples_split=5, min_samples_leaf=7,␣
 ↪max_depth=5)
reg1 = RandomForestRegressor(random_state=0, bootstrap=True, max_features=1.0,
                                **{'n_estimators': 150, 'min_samples_split': 15,␣
 ↪'min_samples_leaf': 8,'max_depth': 40,})
reg2 = RandomForestRegressor(random_state=0, bootstrap=True, max_features=1.0,
                                **{'n_estimators': 250, 'min_samples_split': 15,␣
 ↪'min_samples_leaf': 12,'max_depth': 25,})
reg3 = RandomForestRegressor(random_state=0, bootstrap=True, max_features=1.0,
                                **{'n_estimators': 300, 'min_samples_split': 10,␣
 ↪'min_samples_leaf': 12, 'max_depth': 20,})

if(RSCV==False):
    rf_tas = reg0.fit(X_train_tas, y_train_tas)
    # rf_pr = reg1.fit(X_train_pr, y_train_pr)
    # rf_pr90 = reg2.fit(X_train_pr90, y_train_pr90)
    # rf_dtr = reg3.fit(X_train_dtr, y_train_dtr)
else:
    rf_random0 = RandomizedSearchCV(estimator = reg0, param_distributions =␣
 ↪random_grid, n_iter = 9, cv = 3, verbose=2, n_jobs = -1)
    # rf_random1 = RandomizedSearchCV(estimator = reg1, param_distributions =␣
 ↪random_grid, n_iter = 9, cv = 3, verbose=2, n_jobs = -1)
    # rf_random2 = RandomizedSearchCV(estimator = reg2, param_distributions =␣
 ↪random_grid, n_iter = 9, cv = 3, verbose=2, n_jobs = -1)
```

```
    # rf_random3 = RandomizedSearchCV(estimator = reg3, param_distributions =␣
↪random_grid, n_iter = 9, cv = 3, verbose=2, n_jobs = -1)

    #n_iter = 29

    rf_tas = rf_random0.fit(X_train_tas, y_train_tas)
    # rf_pr = rf_random1.fit(X_train_pr, y_train_pr)
    # rf_pr90 = rf_random2.fit(X_train_pr90, y_train_pr90)
    # rf_dtr = rf_random3.fit(X_train_dtr, y_train_dtr)
```

[ ]:
```
if(RSCV==True):
    print(rf_tas.best_params_)
    # print(rf_pr.best_params_)
    # print(rf_pr90.best_params_)
    # print(rf_dtr.best_params_)
```

```
{'n_estimators': 200, 'min_samples_split': 25, 'min_samples_leaf': 4,
'max_features': None, 'max_depth': 50, 'bootstrap': True}
```

[ ]:
```
## Test on SSP245

test_Y = xr.open_dataset(data_path + 'test/outputs_ssp245.nc').compute()
test_X = xr.open_dataset(data_path + 'test/inputs_ssp245.nc').compute()

tas_truth = test_Y["tas"].mean('member')
pr_truth = test_Y["pr"].mean('member') * 86400
pr90_truth = test_Y["pr90"].mean('member') * 86400
dtr_truth = test_Y["diurnal_temperature_range"].mean('member')

test_inputs = pd.DataFrame({
    "CO2": normalize_co2(test_X["CO2"].data),
    "CH4": normalize_ch4(test_X["CH4"].data)
}, index=test_X["CO2"].coords['time'].data)

### Combine with aerosol EOFs
test_inputs=pd.concat([test_inputs,
                       bc_solver.projectField(test_X["BC"], neofs=5,␣
↪eofscaling=1).to_dataframe().unstack('mode').rename(columns={i:f"BC_{i}" for␣
↪i in range(5)}),
                       so2_solver.projectField(test_X["SO2"], neofs=5,␣
↪eofscaling=1).to_dataframe().unstack('mode').rename(columns={i:f"_{i}" for i␣
↪in range(5)}),
                       ], axis=1)
```

[ ]:
```
test_inputs
```

[ ]:

	CO2	CH4	(pseudo_pcs, BC_0)	(pseudo_pcs, BC_1)	\
2015	0.161692	0.467171	2.706810	-1.375172	
2016	0.165480	0.467151	2.648897	-1.303500	
2017	0.169305	0.467130	2.590985	-1.231828	
2018	0.173166	0.467110	2.533072	-1.160157	
2019	0.177064	0.467090	2.475159	-1.088485	
…	…	…	…	…	
2096	0.469755	0.352172	0.434483	-0.189906	
2097	0.471474	0.351729	0.422541	-0.181722	
2098	0.473128	0.351287	0.410599	-0.173539	
2099	0.474718	0.350845	0.398656	-0.165355	
2100	0.476242	0.350403	0.386714	-0.157171	

	(pseudo_pcs, BC_2)	(pseudo_pcs, BC_3)	(pseudo_pcs, BC_4)	\
2015	-0.819380	-0.256301	-1.716714	
2016	-0.707064	-0.314685	-1.616016	
2017	-0.594749	-0.373069	-1.515319	
2018	-0.482433	-0.431453	-1.414621	
2019	-0.370117	-0.489837	-1.313923	
…	…	…	…	
2096	-0.115759	-0.230166	0.147331	
2097	-0.101978	-0.213992	0.169751	
2098	-0.088197	-0.197818	0.192172	
2099	-0.074416	-0.181644	0.214592	
2100	-0.060636	-0.165471	0.237013	

	(pseudo_pcs, _0)	(pseudo_pcs, _1)	(pseudo_pcs, _2)	(pseudo_pcs, _3)	\
2015	0.903505	2.331736	-0.104266	-1.138111	
2016	0.857117	2.261433	0.078151	-1.083139	
2017	0.810728	2.191129	0.260569	-1.028166	
2018	0.764340	2.120826	0.442987	-0.973193	
2019	0.717951	2.050522	0.625405	-0.918221	
…	…	…	…	…	
2096	0.178097	0.697813	1.189839	-0.064685	
2097	0.177501	0.687329	1.167448	-0.080802	
2098	0.176905	0.676844	1.145056	-0.096920	
2099	0.176309	0.666359	1.122665	-0.113037	
2100	0.175713	0.655874	1.100274	-0.129154	

	(pseudo_pcs, _4)
2015	1.045259
2016	0.977491
2017	0.909724
2018	0.841956
2019	0.774188
…	…
2096	0.033707

```
2097          0.006906
2098         -0.019895
2099         -0.046696
2100         -0.073497

[86 rows x 12 columns]
```

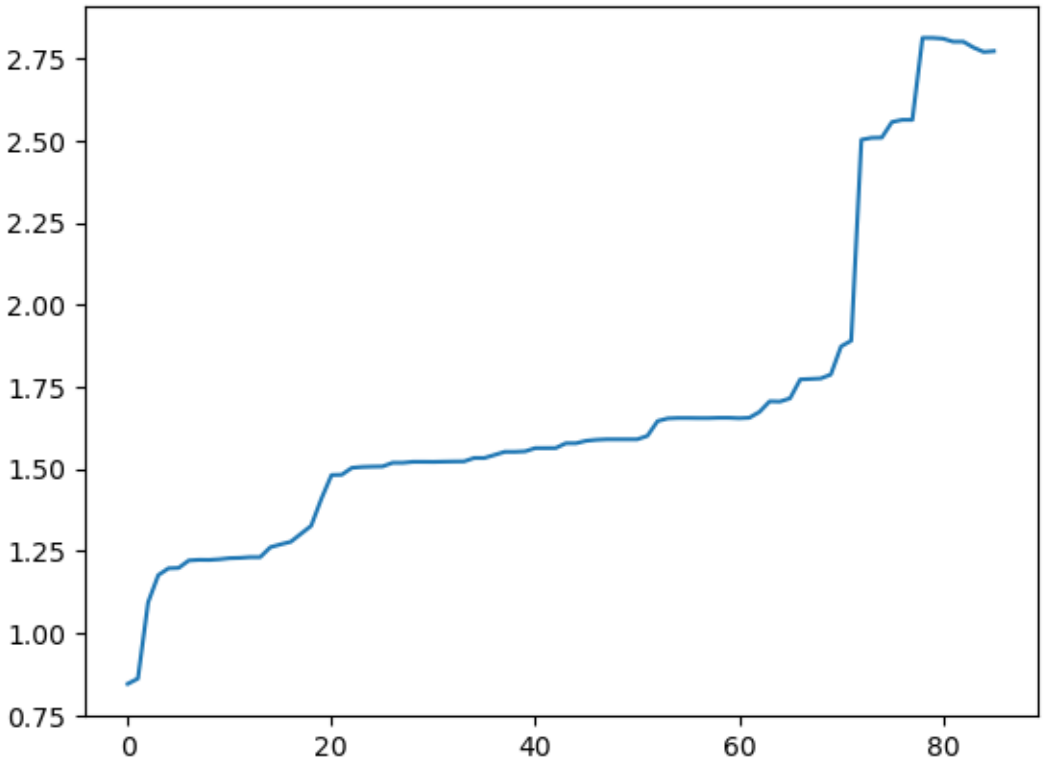
[ ]:
```
rf_tas
```

[ ]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(random_state=0),
                   n_iter=9, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [5, 10, 15, 20, 25, 30, 35,
                                                      40, 45, 50, 55, None],
                                        'max_features': ['log2', 'sqrt', None],
                                        'min_samples_leaf': [4, 8, 12],
                                        'min_samples_split': [5, 10, 15, 25],
                                        'n_estimators': [100, 150, 200, 250,
                                                         300]},
                   verbose=2)
```

[ ]:
```
m_out_t = rf_tas.predict(test_inputs.values)
# m_out_p = rf_pr.predict(test_inputs.values)
# m_out_p90 = rf_pr90.predict(test_inputs.values)
# m_out_d = rf_dtr.predict(test_inputs.values)

m_out_tas = m_out_t.reshape(86, 96, 144)
# m_out_pr = m_out_p.reshape(86, 96, 144)
# m_out_pr90 = m_out_p90.reshape(86, 96, 144)
# m_out_dtr = m_out_d.reshape(86, 96, 144)
```

[ ]:
```
plt.plot(m_out_t.mean(axis=-1))
```

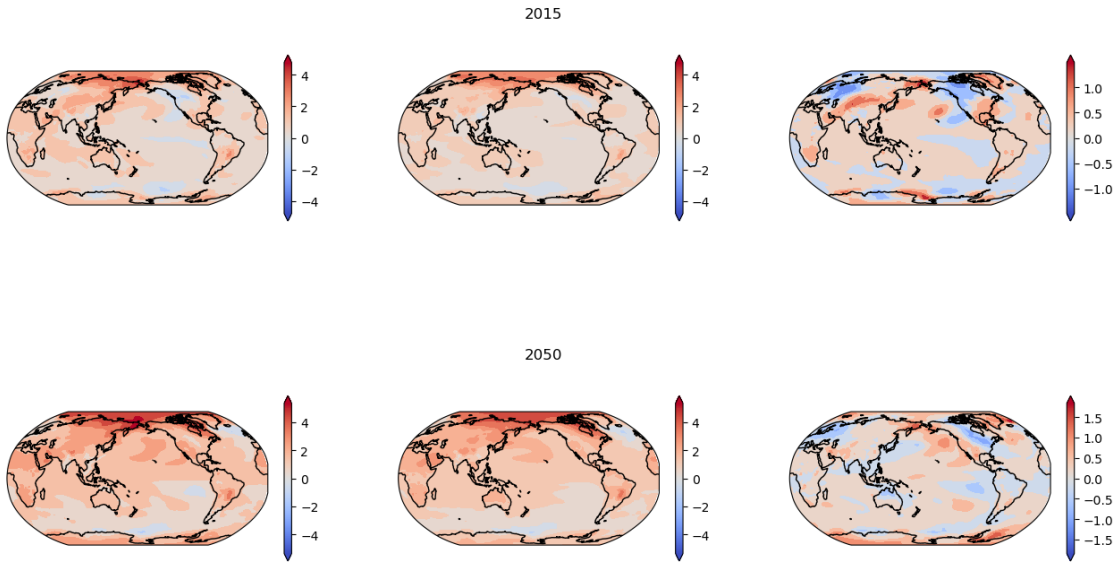
[ ]:
```
[<matplotlib.lines.Line2D at 0x155331b48be0>]
```

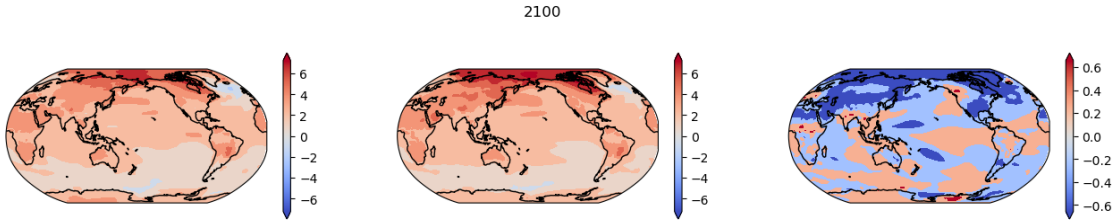


[ ]:
```
xr_output=xr.Dataset(coords={'time': test_X.time.values, 'lat': test_X.latitude.
 ↪values, 'lon': test_X.longitude.values})
xr_output["tas"]=(['time', 'lat', 'lon'],  m_out_tas)
# xr_output["diurnal_temperature_range"]=(['time', 'lat', 'lon'],  m_out_dtr)
# xr_output["pr"]=(['time', 'lat', 'lon'],  m_out_pr)
# xr_output["pr90"]=(['time', 'lat', 'lon'],  m_out_pr90)
```

[ ]:
```
norm_diff = colors.TwoSlopeNorm(vmin=-2, vcenter=0, vmax=2)
cmap = 'coolwarm'
for i in [0, 35, 85]:
    fig = plt.figure(figsize=(16, 3))
    ax = fig.add_subplot(131, projection=cartopy.crs.
 ↪Robinson(central_longitude=180))
    x = tas_truth.isel(time=i)
    x_max = np.max(x)
    norm = colors.TwoSlopeNorm(vmin=-np.abs(x_max), vcenter=0, vmax=np.
 ↪abs(x_max))
    ax.contourf(tas_truth.lon, tas_truth.lat, x, transform=cartopy.crs.
 ↪PlateCarree(),
                norm=norm, cmap=cmap)
    ax.add_feature(cartopy.feature.COASTLINE)
    fig.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
```

```
            ax=ax, orientation='vertical', extend='both', shrink=.8)

  ax = fig.add_subplot(132, projection=cartopy.crs.
↪Robinson(central_longitude=180))
  x = xr_output['tas'].isel(time=i)
  ax.contourf(tas_truth.lon, tas_truth.lat, x, transform=cartopy.crs.
↪PlateCarree(),
              norm=norm, cmap=cmap)
  ax.add_feature(cartopy.feature.COASTLINE)
  plt.suptitle(str(2015+i))
  fig.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
            ax=ax, orientation='vertical', extend='both', shrink=.8)
  ax = fig.add_subplot(133, projection=cartopy.crs.
↪Robinson(central_longitude=180))
  x = tas_truth.isel(time=i)-xr_output['tas'].isel(time=i)
  x_max = np.max(x)
  norm = colors.TwoSlopeNorm(vmin=-np.abs(x_max), vcenter=0, vmax=np.
↪abs(x_max))
  ax.contourf(tas_truth.lon, tas_truth.lat, x, transform=cartopy.crs.
↪PlateCarree(),
              norm=norm, cmap=cmap)
  fig.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
            ax=ax, orientation='vertical', extend='both', shrink=.8)
  ax.add_feature(cartopy.feature.COASTLINE)
  plt.show()
```



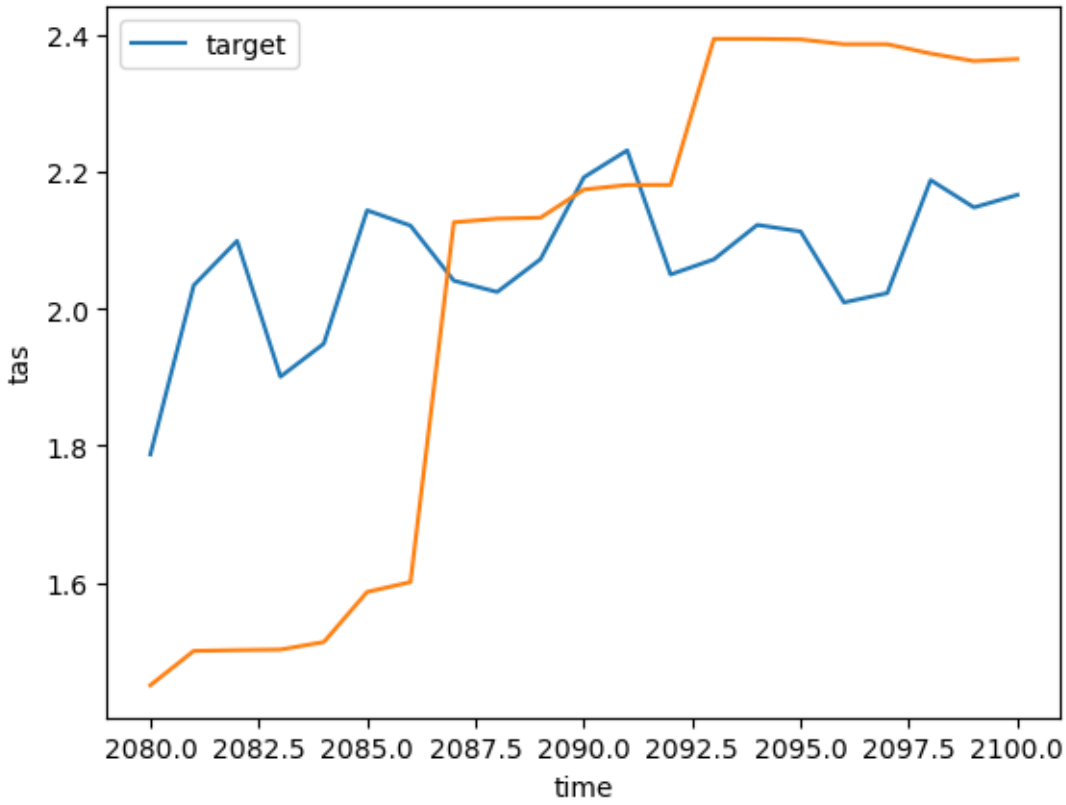


[ ]:
```
variable = 'tas'
start = 2080
weights = np.cos(np.deg2rad(Y['tas'].lat)).expand_dims(lon=144).
 ↪assign_coords(lon=Y.lon)
nrmse = rmse(tas_truth.sel(time=slice(start, None)).mean('time'),
            xr_output[variable].sel(time=slice(start, None)).mean('time'),␣
 ↪weights=weights).data/ np.abs(
               global_mean(tas_truth.sel(time=slice(start, None)).
 ↪mean('time')).data)
print(nrmse)
r2e = rmse(global_mean(tas_truth.sel(time=slice(start, None))),
           global_mean(xr_output[variable].sel(time=slice(start, None)))).data/␣
 ↪np.abs(
              global_mean(tas_truth.sel(time=slice(start, None)).mean('time')).
 ↪data)
print(r2e)
print(nrmse+5*r2e)
```

```
0.08411859630678128
0.162332797972665
0.8957825861701063
```

[ ]:
```
global_mean(tas_truth.sel(time=slice(start, None))).plot(label='target')
global_mean(xr_output[variable].sel(time=slice(start, None))).plot()
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x15532a191c40>
```



[ ]:
```
print(f"RMSE: {get_rmse(tas_truth[35:], m_out_tas[35:]).mean()}")
print("\n")
```

```
RMSE: 0.5244521282584569
```

[ ]:
```

```