```python
#!/usr/bin/python3
import numpy as np
import pandas as pd
import math
from sklearn import tree
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import BayesianRidge
import pickle
import matplotlib.pyplot as plt


def calcAngle(o_leg, a_leg):
    'calculate wind angle from u_10 and v_10 data'
    if o_leg > 0 and a_leg > 0:
        o_leg = abs(o_leg)
        a_leg = abs(a_leg)
        return math.degrees(math.atan(o_leg / a_leg)) + 180

    if o_leg < 0 and a_leg < 0:
        o_leg = abs(o_leg)
        a_leg = abs(a_leg)
        return math.degrees(math.atan(o_leg / a_leg))

    if o_leg < 0 and a_leg > 0:
        o_leg = abs(o_leg)
        a_leg = abs(a_leg)
        return 180 - math.degrees(math.atan(o_leg / a_leg))

    if o_leg > 0 and a_leg < 0:
        o_leg = abs(o_leg)
        a_leg = abs(a_leg)
        return 360 - math.degrees(math.atan(o_leg / a_leg))

def calcWind(u, v):
    'calculate wind speed from u_10 and v_10 data'
    speed = np.array((u * u + v * v).pow(0.5))
    helper = np.vectorize(calcAngle)
    direction = helper(u, v)
    return np.array([speed, direction])

def calcRh(d, t):
    return np.array(100*(np.exp((d * 17.625) / (234.04 + d)) /
                           (np.exp((t * 17.625) / (234.04  + t)))))

def formatDate(dataframe):
    dataframe["forecast_date"] = str(dataframe["time"]).split(" ")[0]
    dataframe["horizon"] = str(dataframe["time"]).split(" ")[1].split(":")[0]  + "hour"

def interpolate(frames):
    'linearly interpolate values for every full hour'
    newFrames = []
    for ele in frames:
        insert = 6
        ele.index = range(0, (len(ele)) * insert, insert)
        ele = ele.reindex(index=range((len(ele)-1)*insert + 1))
        ele = ele.interpolate()
        newFrames.append(ele)
    surInterpolated = pd.concat(newFrames, ignore_index=True)
    print(surInterpolated.shape)
    return surInterpolated

def splitInTwenty(surface):
    'split data into packs of twenty, separating the 50 ensembles'
    frames = []
    for i in range(int(len(surface)/20)):
        frames.append(surface[i*20:i*20+20:1])
    return frames

def accumulateTp(precip):
    'accumulate synop precipitation to match 6hr format'
    accumulated = np.array(precip.rolling(6).sum())
    return accumulated

def createTrainingSetTimeSeries(input, target):
    'create data frame with training data points consisting of time series of three'
    oneTrainingInput=[]
    oneTarget=[]

    for frame in input:
        for i in range(len(frame)):
            if i > 0 and i < len(frame) - 1:
                oneTrainingInput.append(list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i-1])
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i])
```

```python
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i+1]))
            elif i == 0:
                oneTrainingInput.append(list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i])
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i])
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i+1]))
            else:
                oneTrainingInput.append(list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i-1])
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i])
                + list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i]))
            time = frame.valid_time.iloc[i]
            correctTarget = target[target.datetime == time][["temp", "wind_direction", "wind_speed", "precip_quantity_6hr"]]
            oneTarget.append(list(correctTarget.iloc[0]))
    return pd.DataFrame(list(zip(oneTrainingInput, oneTarget)), columns=["Input", "Target"])

def createTrainingSet(input, target):
    'create dataframe with training data points'
    oneTrainingInput=[]
    oneTarget=[]
    for frame in input:
        for i in range(len(frame)):
            oneTrainingInput.append(list(frame[["t2m", "wind_direction", "wind_speed", "tp6"]].iloc[i]))
            time = frame.valid_time.iloc[i]
            correctTarget = target[target.datetime == time][["temp", "wind_direction", "wind_speed", "precip_quantity_6hr"]]
            oneTarget.append(list(correctTarget.iloc[0]))
    return pd.DataFrame(list(zip(oneTrainingInput, oneTarget)), columns=["Input", "Target"])

def matchTraining(input):
    'match inference input data with training data'
    sets = []
    oneInput = []
    for i in range(50):
        sets.append(input[i::50])
    for set in sets:
        for i in range(len(set)):
            if i > 0 and i < len(set) - 1:
                oneInput.append(list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i-1])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i+1]))
            elif i == 0:
                oneInput.append(list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i+1]))
            else:
                oneInput.append(list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i-1])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i])
                + list(set[["t2m", "wind_direction", "wind_speed", "tp"]].iloc[i]))
    return pd.DataFrame(oneInput)

def treeRegressor(input):
    'loading and applying best tree model during inference'
    temperatureTree = pickle.load(open("tempTree", "rb"))
    windTree = pickle.load(open("windTree", "rb"))
    precipTree = pickle.load(open("precipTree", "rb"))
    predTemp = temperatureTree.predict(input)
    predWind = windTree.predict(input)
    predPrecip = precipTree.predict(input)
    predictions = pd.DataFrame(columns=["t2m", "wind", "precip"])
    predictions["t2m"] = predTemp
    predictions["wind"] = predWind
    predictions["precip"] = predPrecip
    return predictions

def forestRegressor(input):
    'loading and applying best forest model during inference'
    tempForest = pickle.load(open("tempForest", "rb"))
    windForest = pickle.load(open("windForest", "rb"))
    precipForest = pickle.load(open("precipForest", "rb"))
    predTemp = tempForest.predict(input)
    print(predTemp[::20])
    print(len(predTemp[::20]))
    predWind = windForest.predict(input)
    predPrecip = precipForest.predict(input)
    predictions = pd.DataFrame(columns=["t2m", "wind", "precip"])
    predictions["t2m"] = predTemp
    predictions["wind"] = predWind
    predictions["precip"] = predPrecip
    return predictions

def quantiles(predictions):
    'calculate quantiles for desired output during inference'
    result = pd.DataFrame(columns=["forecast_date", "target", "horizon", "q0.025", "q0.25", "q0.5", "q0.75", "q0.975"])
    quants = [0.025, 0.25, 0.5, 0.75, 0.975]
    for i in range(0, 20):
        horizon = i * 6 + 6
        quantsTimestepTemp = []
        quantsTimestepWind = []
        quantsTimestepPrecip = []
```

```python
        for quant in quants:
            quantsTimestepTemp.append(np.quantile(predictions["t2m"][i::20], quant))
            quantsTimestepWind.append(np.quantile(predictions["wind"][i::20], quant))
            quantsTimestepPrecip.append(np.quantile(predictions["precip"][i::20], quant))
        result = result.append(pd.Series(["2023-05-06", "t2m", str(horizon) + " hour", *quantsTimestepTemp],
                                         index=["forecast_date", "target", "horizon", "q0.025", "q0.25", "q0.5", "q0.75", "q0.975"]),
                                         ignore_index=True)
        result = result.append(pd.Series(["2023-05-06", "wind", str(horizon) + " hour", *quantsTimestepWind],
                               index=["forecast_date", "target", "horizon", "q0.025", "q0.25", "q0.5", "q0.75", "q0.975"]),
                               ignore_index=True)
        result = result.append(pd.Series(["2023-05-06", "precip", str(horizon) + " hour", *quantsTimestepPrecip],
                               index=["forecast_date", "target", "horizon", "q0.025", "q0.25", "q0.5", "q0.75", "q0.975"]),
                               ignore_index=True)
    return result

def cross_validation(model, input, target):
    'perform k-fold cross validation, return training and validation rmse'
    kf = KFold(5, shuffle = True, random_state = 0)
    rmse = [[], []]
    for train_ind, val_ind in kf.split(input, target):
        model.fit(input.iloc[train_ind], target.iloc[train_ind])
        rmse[0].append(mean_squared_error(model.predict(input.iloc[train_ind]), target.iloc[train_ind])**0.5)
        rmse[1].append(mean_squared_error(model.predict(input.iloc[val_ind]), target.iloc[val_ind])**0.5)
    return [np.mean(rmse[0]), np.mean(rmse[1])]

def validation(model, input, target):
    'perform validation of a model, return training and validation rmse'
    xTrain, xVal, yTrain, yVal = train_test_split(input, target, test_size=0.2, random_state = 0, shuffle=True)
    model.fit(xTrain, yTrain)
    rmse_train = mean_squared_error(model.predict(xTrain), yTrain)**0.5
    rmse_val = mean_squared_error(model.predict(xVal), yVal)**0.5
    return [rmse_train, rmse_val]

def trees(xTrain, yTrain):
    'experimental setup to find the best depth for the tree of all three target variables'
    start = 5
    end = 15
    temp = []
    wind = []
    precip = []
    for depth in range(start, end):
        temperatureTree = tree.DecisionTreeRegressor(max_depth=depth)
        windTree = tree.DecisionTreeRegressor(max_depth=depth)
        precipTree = tree.DecisionTreeRegressor(max_depth=depth)
        temp.extend(cross_validation(temperatureTree, xTrain, yTrain[[0]]))
        wind.extend(cross_validation(windTree, xTrain, yTrain[[2]]))
        precip.extend(cross_validation(precipTree, xTrain, yTrain[[3]]))
    fig, axis = plt.subplots(1, 3)
    axis[0].plot(range(start, end), temp[::2])
    axis[0].plot(range(start, end), temp[1::2])
    axis[0].set_title("Temperature")
    axis[0].set_xlabel("Depth of tree")
    axis[0].set_ylabel("RMSE")

    axis[1].plot(range(start, end), wind[::2])
    axis[1].plot(range(start, end), wind[1::2])
    axis[1].set_title("Wind speed")
    axis[1].set_xlabel("Depth of tree")

    axis[2].plot(range(start, end), precip[::2])
    axis[2].plot(range(start, end), precip[1::2])
    axis[2].set_title("Precipitation")
    axis[2].set_xlabel("Depth of tree")

    axis[0].grid()
    axis[1].grid()
    axis[2].grid()
    plt.savefig("test.pdf", format="pdf", bbox_inches="tight")
    plt.show()
    # best trees
    # temp: 11
    # wind: 13
    # precip: 8

def randomForest(xTrain, yTrain):
    'experimental setup to find the best number of trees in the forest for all three target variables'
    start = 5
    end = 100
    temp = []
    wind = []
    precip = []
    for number_trees in range(start, end, 10):
        tempForest = RandomForestRegressor(number_trees, max_depth=11)
        windForest = RandomForestRegressor(number_trees, max_depth=13)
        precipForest = RandomForestRegressor(number_trees, max_depth=7)
        temp.extend(validation(tempForest, xTrain, yTrain[[0]]))
        wind.extend(validation(windForest, xTrain, yTrain[[2]]))
```

```
        precip.extend(validation(precipForest, xTrain, yTrain[[3]]))
    fig, axis = plt.subplots(1, 3)
    axis[0].plot(range(start, end, 10), temp[::2])
    axis[0].plot(range(start, end, 10), temp[1::2])
    axis[0].set_title("Temperature")
    axis[0].set_xlabel("Number of trees")
    axis[0].set_ylabel("RMSE")

    axis[1].plot(range(start, end, 10), wind[::2])
    axis[1].plot(range(start, end, 10), wind[1::2])
    axis[1].set_title("Wind speed")
    axis[1].set_xlabel("Number of trees")

    axis[2].plot(range(start, end, 10), precip[::2])
    axis[2].plot(range(start, end, 10), precip[1::2])
    axis[2].set_title("Precipitation")
    axis[2].set_xlabel("Number of trees")

    axis[0].grid()
    axis[1].grid()
    axis[2].grid()
    plt.savefig("randomForest.pdf", format="pdf", bbox_inches="tight")
    plt.show()
    # temp 75
    # wind 85
    # precip 35

def neuralNet(xTrain, yTrain):
    'experimental setup to find the best neural net architecture'
    dimensions = [[12, 9, 6], [12, 8]]
    losses = []
    validation = []
    for dimension in dimensions:
        neuralNet = MLPRegressor(dimension, early_stopping=True)
        neuralNet.fit(xTrain, yTrain[[0, 2, 3]])
        losses.append(list(map(lambda x: x**0.5, neuralNet.loss_curve_)))
        validation.append(neuralNet.validation_scores_)
    fig, axis = plt.subplots(2, len(dimensions))
    for i in range(len(dimensions)):
        axis[i, 0].plot(losses[i])
        axis[i, 0].set_title(f"Loss Architecture " + str(dimensions[i]))
        axis[i, 0].set_xlabel("Epochs")
        axis[i, 0].set_ylabel("MSE")
        axis[i, 0].grid()
        axis[i, 1].plot(validation[i])
        axis[i, 1].set_title(f"Validation score Architecture " + str(dimensions[i]))
        axis[i, 1].set_xlabel("Epochs")
        axis[i, 1].set_ylabel("R2 score")
        axis[i, 1].grid()
    plt.savefig("neuralNet2.pdf", format="pdf")
    plt.show()
    # second architecture performs better
    # stopping at 18 iterations

def bayRidge(xTrain, yTrain):
    'experimental setup to find the best number of iterations for all three bayesian ridge models'
    start = 1
    end = 100
    temp = []
    wind = []
    precip = []
    for iters in range(start, end, 10):
        temperatureBay = BayesianRidge(n_iter=iters)
        windBay = BayesianRidge(n_iter=iters)
        precipBay = BayesianRidge(n_iter=iters)
        temp.extend(cross_validation(temperatureBay, xTrain, yTrain[[0]]))
        wind.extend(cross_validation(windBay, xTrain, yTrain[[2]]))
        precip.extend(cross_validation(precipBay, xTrain, yTrain[[3]]))
    fig, axis = plt.subplots(1, 3)
    axis[0].plot(range(start, end, 10), temp[::2])
    axis[0].plot(range(start, end, 10), temp[1::2])
    axis[0].set_title("Temperature")
    axis[0].set_xlabel("No. iterations")
    axis[0].set_ylabel("RMSE")

    axis[1].plot(range(start, end, 10), wind[::2])
    axis[1].plot(range(start, end, 10), wind[1::2])
    axis[1].set_title("Wind speed")
    axis[1].set_xlabel("No. iterations")

    axis[2].plot(range(start, end, 10), precip[::2])
    axis[2].plot(range(start, end, 10), precip[1::2])
    axis[2].set_title("Precipitation")
    axis[2].set_xlabel("No. iterations")

    axis[0].grid()
    axis[1].grid()
```

```
    axis[2].grid()
    plt.savefig("bayRidge.pdf", format="pdf", bbox_inches="tight")
    plt.show()
```