Sequential Now-fasting: To be used in Early Warning Systems.

import numpy as np

import pandas as pd

import pickle

import pdb

from sklearn.ensemble import RandomForestRegressor as RF

from sklearn.metrics import r2\_score as r2

from sklearn.metrics import mean\_squared\_error as mse

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Ridge as ridge

Setting the seed ensures that the results will be the same when reproduced.

*##set seed*

np.random.seed(9823)

Setting the Forest Size to 2000

nt = 2000

Creating a vector that has different random forest methods: independent(ind) or joint

rftypes = ['ind','joint'] *##ind must be first*

Setting the countries, these might be different for you.

countries = ["Bangladesh","Ethiopia","Ghana","Kenya","Mali","Nepal","Nigeria","Senegal","Uganda"]

Setting the years, these might be different for you.

pyears = [["04","07","11","14"],["05","11","16"],["08","14"],['08','14'],["06","12"],["06","11","16"],["08","13"],["05",'10'],["06",'11','16']]

Setting the different possible outcomes, these might be different for you.

outcomes = ['stunted', 'wasted',  'healthy', 'poorest','underweight\_bmi']

For loop to sort the years into a list.

syears = np.sort(list(set([y for years in pyears for y in years])))

Create a pandas dataframe using the data that was cleaned by Linden McBride.

xa = pd.read\_csv('data/data.csv')

Imps is used to produce some figures in the paper and is not needed for predictions

imps = [[] for country in countries]

*##for each survey year, for each country*

for i,year in enumerate(syears):

    for j,country in enumerate(countries):

 Prints the index of the current syears and prints index of the current countries

        print(i,j)

        *##filter data*

Only keep countries that we’re looking at, these might be different for you.

        x = xa[xa.country == country]

        x = x.drop('country',axis=1)

 Set the year for x. We do 20 + year because year only contains the last 2 digits of the year.

        x = x[x.year <= int('20' + year)]

Split the data into Training(xtr) and Testing(xte) data

        *##test train split*

Training data is when the data’s year is less than the specified year.

        xtr = x[x.year < int('20' + year)]

        *##sequential*

Testing Data is the data with the specified year.

        xte = x[x.year == int('20' + year)]

You will not have to do this specific data handling depending on your dataset.

        *##handles senegal 12*

        if (np.shape(xtr)[0]\*np.shape(xte)[0]>0)\*(not country == 'Senegal' or not year == '12'):

 The below code chunk takes out data where more than 20% of the data is missing. Here it is just taking out the market price data. You may need to shift this number according to your dataset.

            *##drop largely missing features*

            xf = pd.concat([xtr,xte],ignore\_index=True)

            mask = np.array(xf.isnull().sum()/len(x))

            xtr = xtr.loc[:,mask<.2]

            xte = xte.loc[:,mask<.2]

 Split the data into Training and Testing Data

            *##drop rows with missing data*

 Drop x training data that are NA

            xtr = xtr.dropna(axis=0)

 Drop x testing data that are NA

            xte = xte.dropna(axis=0)

            *##make random forests*

            g=dict({'max\_depth':4,'max\_features':.333,'bootstrap':True})

 Labs is created because xtr contains the predictions at the end of the list, but when recreating this, we suggest keeping the predictions in a separate variable to minimize confusion. Additionally, the country is in the 0th column and we need to omit this when making the predictions, hence why labs starts at the 1st column.

            labs = list(xtr.columns)[1:-5]

 Joint Random Forest

            for rftype in rftypes:

                if rftype =='joint':

 Load the data into a numpy data frame called W

                    W = np.load('data/W/W'+year+country+'seq.npy')

 Create a variable ytr with the y training data by taking the dot product of W and the transpose of xtr[outcomes] and the transpose of that

                    ytr = np.dot(W,xtr[outcomes].T).T

 Create a variable yte with the y testing data by taking the dot product of W and the transpose of xte[outcomes] and the transpose of that

                    yte = np.dot(W,xte[outcomes].T).T

 Create the random Forests

                    rf = RF(nt,max\_depth=4,max\_features = .333)

                    rf = rf.fit(xtr[labs],ytr)

 Set predtr(prediction training) as the prediction of xtr[labs](x testing random forest) from the given decision trees.

                    predtr =  rf.predict(xtr[labs])

 Set predte(prediction testing) as the prediction of xte[labs](x training random forest) from the given decision trees.

                    predte = rf.predict(xte[labs])

Sklearn does not support using the Random forest method on dependent data. Below, a whitening technique is used to make the data independent by orthogonalizing the data so the random forest technique can be used.

 Set Winv as the inverse of W

                    Winv = np.linalg.inv(W)

 Set predtr as the transpose of the dot product of Winv and predtr.T(transpose of predtr)

                    predtr = np.dot(Winv,predtr.T).T

 Set predte as the transpose of the dot product of Winv and predte.T(transpose of predte)

                    predte = np.dot(Winv,predte.T).T

 Append the important features from the decision trees to imps[j]. This variable is used to produce some figures but not used to make predictions.

                    imps[j].append(rf.feature\_importances\_)

 For loop of the outcomes to set each respective outcome

                    for k,outcome in enumerate(outcomes):

 Set the outcome + 'rf'+ joint = the predicted outcome found above in xtr

                        xtr[outcome+'rf' + rftype] = predtr[:,k]

 Set the outcome +'rf' + joint = the predicted outcome found above in xte

                        xte[outcome+'rf' + rftype] = predte[:,k]

 Independent Random Forests

                else:

 Set ytr as the training data with the specified outcomes from the top

 ytr = xtr[outcomes]

 Set yte as the testing data with the specified outcomes from the top

                    yte = xte[outcomes]

 For loop through the outcomes.

                    for outcome in outcomes:

This code stores the outcomes in xtrf,/xtef when replicating it is best to create a new variable for outcomes to minimize confusion.

                        rf = RF(nt,max\_depth=4,max\_features = .333)

                        rf = rf.fit(xtr[labs],ytr[outcome])

 Set the x training data to the prediction from the random forests

                        xtr[outcome+'rf' + rftype] = rf.predict(xtr[labs])

 Set the x testing data to the prediction from the random forests

                        xte[outcome+'rf' + rftype] = rf.predict(xte[labs])

 Set W as the correlation of each pair of residuals from the training data’s transposed outcomes, then takes the singular value decomposition and multiply the observation of that matrix which will orthogonalize them and make them independent.

                    W = np.linalg.svd(np.corrcoef((np.array(xtr[outcomes])-np.array(xtr[[outcome + 'rfind' for outcome in outcomes]])).T))[2]

 Save the data file using W set above as a Numpy object.

                    np.save('data/W/W'+year+country+'seq',W)

 Set xtr(x training) with the outcomes of independent and joint random forest of the training data. The results are kept in xte, but you can change this variable so that the results are in a separate variable to minimize confusion.

            xtr = xtr[outcomes + [outcome + 'rfind' for outcome in outcomes] + [outcome + 'rfjoint' for outcome in outcomes]]

 Set xte(x testing) with the outcomes of independent and joint random forest of the testing data. The results are kept in xte, but you can change this variable so that the results are in a separate variable to minimize confusion.

            xte = xte[outcomes + [outcome + 'rfind' for outcome in outcomes] + [outcome + 'rfjoint' for outcome in outcomes]]

 Open the file with the specified year and convert training, testing results, and file as a byte stream.

            with open('results/'+year+country+'resultsseqrf.pkl','wb') as f:

                pickle.dump([xtr,xte],f)