Contemporaneous Forecasting/Prediction:

To be used to inform geographic targeting in poverty or malnutrition interventions or to monitor and evaluate.

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(6023)

Setting the Forest Size

nt = 2000

Five Fold Cross Validation Testing and Training are performed 5 times for each survey

folds = 5

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","Guatemala","Honduras","Mali","Nepal","Kenya","Senegal","Uganda","Nigeria"]

Setting the years, these might be different for you.

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

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 earlier.

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 index of current countries

        print(i,j)

        *##filter data*

        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)]

 if statement states if x is larger than 0, it hasn't finished looking at all the countries yet

        if np.shape(x)[0]>0:

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*

      mask = np.array(x.isnull().sum()/len(x)) The average when x is null into mask.

            x = x.loc[:,mask<.2] Locates the index of x.

 Drops NA rows.

            *##drop rows with missing data*

            x = x.dropna(axis=0)

            *##test train split and CV*

Set the x training data when the year is greater than the x year.

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

 Set the x year when it’s equal to year.

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

 Keep doing this when there’s still information in x(x is greater than 0)

            if np.shape(x)[0]>0:

 Set q as the vector of x[0] from the first element to the second to last.

                q = np.arange(np.shape(x)[0])

 Cross Validation

 It uses 5 overlapping subsets, q indices of x, and then shuffles the data and 80% of the data.

                for fold in range(folds):

 Shuffle the countries randomly

                    np.random.shuffle(q)

                    trinds = q[:int(.8\*len(q))]

                    teinds = q[int(.8\*len(q)):]

                    xtrf = pd.concat([xtr,x.iloc[trinds,:]],ignore\_index=True)

                    xtef = x.iloc[teinds,:]

                    *##make random forests*

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(xtrf.columns)[1:-5]

                    for rftype in rftypes:

 Joint Random Forest

                        if rftype =='joint':

 Load the data into a numpy data frame called W

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

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

                            ytrf = np.dot(W,xtrf[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

                            ytef = np.dot(W,xtef[outcomes].T).T

 Create the Random Forests.

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

                            rf = rf.fit(xtrf[labs],ytrf)

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

                            predtr =  rf.predict(xtrf[labs])

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

                            predte = rf.predict(xtef[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):

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

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

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

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

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

 Independent Random Forests

                        else:

 Set ytrf as the training data with the specified outcomes from the top.

                            ytrf = xtrf[outcomes]

 Set ytef as the testing data with the specified outcomes from the top.

                            ytef = xtef[outcomes]

 For loop through the outcomes to set each respective outcome.

                            for outcome in outcomes:

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

                                rf = rf.fit(xtrf[labs],ytrf[outcome])

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

 Set the x training data with the outcomes from the prediction from the random forests.

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

 Set the x testing data with the outcomes from the prediction from the random forests.                                xtef[outcome+'rf' + rftype] = rf.predict(xtef[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(xtrf[outcomes])-np.array(xtrf[[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+'c',W)

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

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

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

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

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

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

                        pickle.dump([xtrf,xtef],f)