faoswsProductionImputation: A package for the imputation of the production domain of the Statistical Working System

Michael. C. J. Kao
Food and Agriculture Organization
of the United Nations

Abstract

This vignette provides detailed description of the usage of functions in the **faoswsProductionImputation** package.

There are three sections to this paper. The opening will describe the essential setups required for the package to operate. This is then followed by the step-by-step description of each function which consist of the whole entire imputation procedure described in the methodology paper. Arguements and default setting for each function is explained with illustrating example. The final section is for technical readers who are interested in building their ensemble model, from how to design sensible component model to how to build the ensemble.

Keywords: Imputation, Linear Mixed Model, Agricultural Production, Ensemble Learning.

DRAFT

1. Setup

Before we begin, we will need to load the required library

```r
## Load libraries
library(faoswsProductionImputation)
library(faoswsExtra)
library(data.table)
library(lattice)
```

To illustrate the functionality of the package, we take the Okra data set for example. The implementation requires the data to be loaded as a *data.table* object. This is also the default when data are queried from the API of the Statistical Working System which we will refer to as SWS from hereon.

```r
str(okrapd)

## Classes 'data.table' and 'data.frame': 912 obs. of  14 variables:
##  $ areaCode          : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ areaName          : chr  "Albania" "Albania" "Albania" "Albania" ...
##  $ itemCode          : int  430 430 430 430 430 430 430 430 430 430 ...
##  $ itemName          : chr  "Okra" "Okra" "Okra" "Okra" ...
##  $ year              : int  1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 ...
##  $ areaHarvestedValue: num  800 600 750 750 740 780 790 780 700 600 ...
##  $ areaHarvestedFlag : chr  "*" "*" "*" "*" ...
##  $ yieldValue        : num  7.25 8 8 8 7.97 ...
##  $ yieldFlag         : chr  "*" "*" "*" "*" ...
##  $ productionValue   : num  5800 4800 6000 6000 5900 6200 6500 6700 6000 5300 ...
##  $ productionFlag    : chr  "*" "*" "*" "*" ...
##  $ productionFlag2   : logi  NA NA NA NA NA NA ...
##  $ areaHarvestedFlag2: logi  NA NA NA NA NA NA ...
##  $ yieldFlag2        : logi  NA NA NA NA NA NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

In addition to the data, the implementation also require a table to map the hierachical relation of the observation flags. In brief, it provides a rule for flag aggregation, an example of the table is given below. For detail treatment and how to create such a table, please see the vignette of the **faoswsFlag** package.

```
swsOldFlagTable = rbind(faoswsFlagTable,
    data.frame(flagObservationStatus = c("*", "F"),
               flagObservationWeights = c(0.9, 0.6)))
swsOldFlagTable[swsOldFlagTable$flagObservationStatus == "E",
                "flagObservationWeights"] = 0.55
swsOldFlagTable

##   flagObservationStatus flagObservationWeights
## 1                                         1.00
## 2                     T                   0.80
## 3                     E                   0.55
## 4                     I                   0.50
## 5                     M                   0.00
## 6                     *                   0.90
## 7                     F                   0.60
```

DRAFT

2. Functions

This section provides the step-by-step usage of functions which are used to perform imputation, the steps illustrated here replicates the one-step imputation function `imputeProductionDomain`.

2.1. Data processing

The first step of the imputation is to remove any previous attempt of imputation. Even for the same methodology and exact setting, prior imputation will vary as more information are received over time. This step is highly recommended but optional and depends on the judgement of the analyst.

To remove the prior imputation, one will need to specify the column name of the value and correspinding flag; further the flag which represents prior imputation and a flag representing missing values. The function will convert the previously imputed value to NA and the flag from previous imputation to a missing flag.

```
okraProcessed = copy(okrapd)

## Removing prior imputation for production
table(okraProcessed$productionFlag)

##
##       *   E   F   M   T
## 487  21  89 150 131  34

removeImputation(data = okraProcessed,
                 value = "productionValue",
                 flag = "productionFlag",
                 imputedFlag = "E",
                 naFlag = "M")
table(okraProcessed$productionFlag)

##
##       *   F   M   T
## 487  21 150 220  34

## Removing prior imputation for area harvested
table(okraProcessed$areaHarvestedFlag)

##
##       *   E   F   M   T
## 430  27 101 188 131  35

removeImputation(data = okraProcessed,
                 value = "areaHarvestedValue",
                 flag = "areaHarvestedFlag",
                 imputedFlag = "E",
                 naFlag = "M")
table(okraProcessed$areaHarvestedFlag)

##
##       *   F   M   T
## 430  27 188 232  35

## Removing prior imputation for yield
table(okraProcessed$yieldFlag)

##
##       *   E   F   M   T
## 410  22 113 200 131  36

removeImputation(data = okraProcessed,
                 value = "yieldValue",
                 flag = "yieldFlag",
                 imputedFlag = "E",
                 naFlag = "M")
table(okraProcessed$yieldFlag)

##
##       *   F   M   T
## 410  22 200 244  36
```

After removing prior imputation, the next step is to replace zero values associating with flag "M" to NA. This is due to the fact that missing values were represented with a value of zero with a flag of "M".

```
removeOM(data = okraProcessed,
         value = "productionValue",
         flag = "productionFlag",
         naFlag = "M")

removeOM(data = okraProcessed,
         value = "areaHarvestedValue",
         flag = "areaHarvestedFlag",
         naFlag = "M")

removeOM(data = okraProcessed,
         value = "yieldValue",
         flag = "yieldFlag",
         naFlag = "M")
```

In order for the linear mixed model to fit successfully, at least one observation is required for each country. Thus, this function removes countries which contains no information or no observation at all.

```
okraProcessed =
    removeNoInfo(data = okraProcessed,
                 flag = "yieldFlag",
                 value = "yieldValue",
                 byKey = "areaCode")
```

The function `processProductionDomain` is a wrapper to execute all the data processing above.

```
okraProcessed =
    processProductionDomain(data = okrapd,
                            productionValue = "productionValue",
                            areaHarvestedValue =
                                "areaHarvestedValue",
                            yieldValue = "yieldValue",
                            yearValue = "year",
                            productionObservationFlag =
                                "productionFlag",
                            areaHarvestedObservationFlag =
                                "areaHarvestedFlag",
                            yieldObservationFlag = "yieldFlag",
                            productionMethodFlag =
                                "productionFlag2",
                            areaHarvestedMethodFlag =
                                "areaHarvestedFlag2",
                            yieldMethodFlag = "yieldFlag2",
                            removePriorImputation = TRUE,
                            removeConflictValues = TRUE,
                            imputedFlag = "E",
```

DRAFT

```
                          naFlag = "M",
                          byKey = "areaCode")
```

2.2. Imputation

Now we are ready to perform the imputation. Recalling the methodology paper, the yield is imputed first. The function `imputeYield` allows the user to specify a desirable formula for the linear mixed model; otherwise the default linear mixed model with spline will be used. If the default model is used, the `maxdf` sets the maximum degree of freedom for the B-spline to be tested. The argument `imputationFlag` and `newMethodFlag` corresponds to the new observation status flag and method flag to be assigned for those values that are imputed.

```
imputeYield(yieldValue = "yieldValue",
            yieldObservationFlag = "yieldFlag",
            yieldMethodFlag = "yieldFlag2",
            yearValue = "year",
            imputationFlag = "I",
            newMethodFlag = "e",
            maxdf = 5,
            byKey = "areaCode",
            data = okraProcessed)
```

After the imputation of yield, we proceed to impute the production.The function `imputeProduction` function actually compose of two steps. During the first step, the entries where the imputed yield can be matched with an existing area harested value are identified, this in turn, enable us to compute the production. If no value for area harvested is available, then the function proceed to impute the remaining production values with ensemble learning. The argument required is largely similar to those of the `imputeYield` function, however, additional parameters are required for the implementation of the ensemble model. A list of **component model** is required, and whether the weights of each component model should be restricted to the specified ceiling for weight allocation. See the next section for more detail.

```
imputeProduction(productionValue = "productionValue",
                 productionObservationFlag = "productionFlag",
                 productionMethodFlag = "productionFlag2",
                 areaHarvestedValue = "areaHarvestedValue",
                 areaHarvestedObservationFlag = "areaHarvestedFlag",
                 yieldValue = "yieldValue",
                 yieldObservationFlag = "yieldFlag",
                 newMethodFlag = "e",
                 data = okraProcessed,
                 restrictWeights = TRUE,
                 maximumWeights = 0.7,
                 byKey = "areaCode",
                 flagTable = swsOldFlagTable)
```

Finally, we can balance the area harvested after both production and yield have been imputed.

```
balanceAreaHarvested(productionValue = "productionValue",
                     productionObservationFlag = "productionFlag",
                     areaHarvestedValue = "areaHarvestedValue",
                     areaHarvestedObservationFlag = "areaHarvestedFlag",
                     areaHarvestedMethodFlag = "areaHarvestedFlag2",
                     yieldValue = "yieldValue",
                     yieldObservationFlag = "yieldFlag",
                     newMethodFlag = "e",
                     data = okraProcessed,
                     flagTable = swsOldFlagTable)
```

The full procedure outlined in this section can be performed by a single function `imputeProductionDomain`.

```
system.time(
    {
        imputedokrapd =
            imputeProductionDomain(data = okrapd,
                                   productionValue = "productionValue",
                                   areaHarvestedValue =
                                       "areaHarvestedValue",
                                   yieldValue = "yieldValue",
                                   yearValue = "year",
                                   productionObservationFlag =
                                       "productionFlag",
                                   areaHarvestedObservationFlag =
                                       "areaHarvestedFlag",
                                   yieldObservationFlag = "yieldFlag",
                                   productionMethodFlag =
                                       "productionFlag2",
                                   areaHarvestedMethodFlag =
                                       "areaHarvestedFlag2",
                                   yieldMethodFlag = "yieldFlag2",
                                   flagTable = swsOldFlagTable,
                                   removePriorImputation = TRUE,
                                   removeConflictValues = TRUE,
                                   imputedFlag = "E",
                                   imputationFlag = "I",
                                   newMethodFlag = "e",
                                   naFlag = "M",
                                   maxdf = 5,
                                   byKey = "areaCode",
                                   restrictWeights = TRUE,
                                   maximumWeights = 0.7)
    }
    )

## Initializing ...
## Imputing Yield ...
## Model with 2 degree of freedom is selected
## Number of values imputed:  195
## Number of values still missing:  0
## Imputing Production ...
## Number of values imputed:  171
## Number of values still missing:  0
## Imputing Area Harvested ...
## Number of values imputed:  194
## Number of values still missing:  0
##    user  system elapsed
##   86.54   91.21   78.62
```

3. Ensemble model

Here we provide some details of how to implement user specific ensemble models.

First of all, the component model need to take a vector of values and return the fitted values. If the model failed or if the fit does not correspond to values in the codomain, then return a vector of NA equal to the length of the input.

Shown below is the default logitstic model in the package, the model will return a vector of NA if there are no observations at both tail. It is the analyst's job to ensure the component models return sensible values. For example, negative values are nonsensical for production, and in the current implementation negative values are replaced with zero.

```
defaultLogistic = function (x){
    time = 1:length(x)
    xmax = max(x, na.rm = TRUE)
    x.scaled = x/xmax
    logisticModel = glm(formula = x.scaled ~ time, family = "binomial")
    logisticFit = predict(logisticModel,
                          newdata = data.frame(time = time),
                          type = "response") * xmax
    midpoint = -coef(logisticModel)[1]/coef(logisticModel)[2]
    if (length(na.omit(x[time < midpoint])) < 1 |
        length(na.omit(x[time > midpoint])) < 1)
        logisticFit = rep(NA, length(x))
    logisticFit
}
```

After defining the component models, the next step is to combine the models into a list. The support functions in the package will then take care of the rest.

```
myModel = list(defaultMean, defaultLm, defaultExp,
        defaultLogistic, defaultLoess, defaultSpline, defaultArima,
        defaultMars, defaultNaive)
```

Here we take the Okra production value of Bahrain as an illustration. After the component models has been designed and inserted into a list, we can first compute the fits and weights then combine it to form the ensemble with the following functions.

```
bahrainExample = okraProcessed[areaName == "Bahrain", productionValue]

## Compute fit for all component models
```

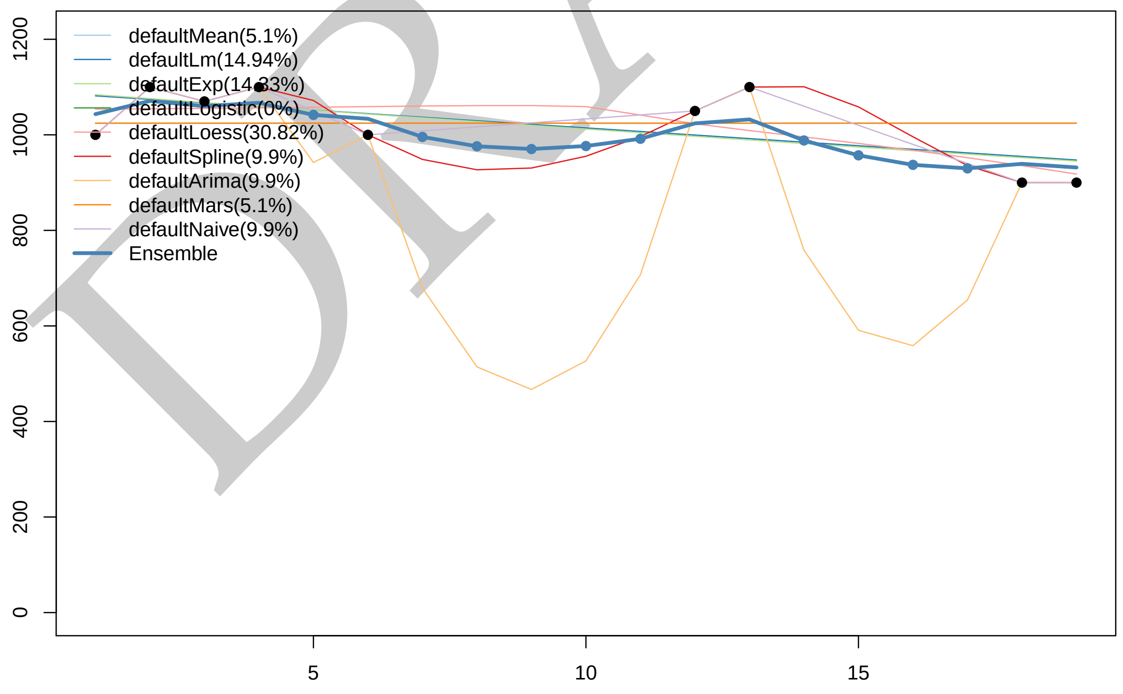
DRAFT

```
modelFits =
    computeEnsembleFit(x = bahrainExample, ensembleModel = myModel)

## Calculate the weight for each component model
modelWeights =
    computeEnsembleWeight(x = bahrainExample,
                          fits = modelFits,
                          restrictWeights = TRUE,
                          maximumWeights = 0.7)

## Combine the models to obtain the ensemble
ensembleFit = computeEnsemble(modelFits, modelWeights)
```

A one-step wrapper function is also available. An optional arguement for ensemble weight is available. You can specify whether to restrict the weights of a single model. In this example, the default restricts the weight and set the maximum weight of a model can take to 0.7.

```
fijiExample = okraProcessed[areaName == "Fiji Islands", productionValue]
ensembleFit =
    ensembleImpute(x = fijiExample,
                   restrictWeights = TRUE,
                   maximumWeights = 0.7,
                   plot = TRUE)
```

DRAFT



Affiliation:

Michael. C. J. Kao
Economics and Social Statistics Division (ESS)
Economic and Social Development Department (ES)
Food and Agriculture Organization of the United Nations (FAO)
Viale delle Terme di Caracalla 00153 Rome, Italy
E-mail: michael.kao@fao.org
URL: https://github.com/mkao006/sws_imputation

DRAFT