Package 'toweranNA'

May 8, 2026

Title A Method for Handling Missing Values in Prediction Applications

Version 0.1.0

Maintainer Norm Matloff <nsmatloff@ucdavis.edu>

Depends R (>= 3.6.0),regtools (>= 0.8.0),rmarkdown

Imports FNN, pdist, stats

License GPL (>= 2)

Description Non-imputational method for handling missing values in a prediction context, meaning that not only are there missing values in the training dataset, but also some values may be missing in future cases to be predicted. Based on the notion of regression averaging (Matloff (2017, ISBN: 9781498710916)).

Encoding UTF-8

URL https://github.com/matloff/toweranNA

BugReports https://github.com/matloff/toweranNA/issues

NeedsCompilation no

Author Norm Matloff [aut, cre] (ORCID: <https://orcid.org/0000-0001-9179-6785>), Pete Mohanty [aut] (ORCID: <https://orcid.org/0000-0001-8531-3345>)

Repository CRAN

Date/Publication 2023-03-15 08:00:02 UTC

Contents

toweranNA-package *toweranNA: a Missing-Values Method Specific to Prediction Applications*

Description

A *nonimputational* method for handling missing values (MVs), specifically for *prediction applications.*

Norm Matloff (UC Davis) and Pete Mohanty (Google)

(This work was performed prior to PM's joining Google, and is not connected to Google in any manner.)

Overview: the Goal Is Prediction, Not Statistical Inference

There are a number of powerful R packages for handling missing values (MVs), such as `Amelia` and `mice`. They "fill in the blanks" in MV-ridden datasets, so as to enable the user to do statistical inference on the completed data.

These methods are typically not capable of predicting new cases that have MVs. With the *toweranNA* package, *the intended class of applications is predictive modeling, rather than estimation.*

Predictive methods of any type can be used with our Tower Method, including both linear/generalized linear models and nonparametric/machine learning methods.

Usage

The function *makeTower* takes the data and regression model as input, and creates an object of class 'tower'. New cases can then be predicted by calling *predict()* on that object.

The call forms are:

```
makeTower(data, yName, regFtnName, opts = NULL, scaling = NULL,
    yesYVal = NULL)
predict(towerObj, newx, k = 1)
```

The main arguments are:

- *data*: The training set, a data frame or equivalent.
- *yName*: Name of the column containing "Y", the variable to be predicted.
- `regFtnName`: Name of the regression model, currently 'lm', 'glm' or 'towerKNN'.
- `opts`: Optional arguments for `regFtnName`.
- `towerObj`: Object of class "tower" returned by `makeTower()`.
- `newx`: The X data to be predicted (one or more new cases).
- `k`, The number of nearest neighbors use in prediction. (Not the same as `kmax` in `towerKNN`.)

The argument *k* is a tuning parameter chosen by the analyst.

Example: Vocabulary Acquisition

This data is from the Stanford University Wordbank project. The data, *english*, is included in the toweranNA package. Of the non-administrative variables, e.g. excluding 'Language', which is always English in this data, about 43 percent of the values are missing.

To illustrate how fitting and prediction occur, let's apply Tower to fit the data and predict Y for the cases having missing values. We will take age and gender as predictors.

```
data(english)

names(english)
#  [1] "data_id"     "age"         "language"    "form"        "birth_order"
#  [6] "ethnicity"   "sex"         "mom_ed"      "measure"     "vocab"
# [11] "demo"        "n"           "demo_label"

# omit administrative variables
engl1 <- english[,c(2,5:8,10)]

head(engl1)
#   age birth_order ethnicity    sex         mom_ed vocab
# 1  24       First     Asian Female       Graduate   337
# 2  19      Second     Black Female        College   384
# 3  24       First     Other   Male Some Secondary    76
# 4  18       First     White   Male      Secondary    19
# 5  24       First     White Female      Secondary   480
# 6  19       First     Other Female   Some College   313

# many MVs
sum(is.na(engl1))
# [1] 9649
# most MVs are near the end
tail(engl1)
#      age birth_order ethnicity    sex mom_ed vocab
# 5493  28        <NA>      <NA>   Male   <NA>   352
# 5494  28        <NA>      <NA> Female   <NA>   460
# 5495  28        <NA>      <NA>   Male   <NA>   292
# 5496  28        <NA>      <NA> Female   <NA>   661
# 5497  28        <NA>      <NA> Female   <NA>   550
# 5498  28        <NA>      <NA>   Male   <NA>   549

# fit linear model for predictingt vocabulary size
towerOut <- makeTower(engl1,'vocab','lm')
```

Say we wish to predict a new case like the child in row 5, but little order, and who is second in birth order, and for whom gender and mother's education are missing.

```
newx <- engl1[5,-6]
```

```
newx$age <- 28
newx$sex <- NA
newx$mom_ed <- NA
newx
#   age birth_order ethnicity sex mom_ed
# 5  28       First     White  NA     NA
predict(towerOut,newx)
# 496.9752
```

toweranNA: A Method Based on Regression Averaging

Setting: We have a dataset in which one of the columns, Y, is to be predicted in the future. The remaining columns, collectively referred to as X, are the predictor variables/features. Y can be either numeric or an R factor.

Most of the MV literature concerns estimation of some relationship, say comparison of means, analysis of linear regression coefficients and the like. One applies some MV method to the original data, obtaining a "filled-in" version of the data (or several such versions). One then performs one's statistical analysis on the new version.

By constrast, our emphasis here is on *PREDICTION*, especially relevant in our AI era. The main contribution of this package is a technique that we call the Tower Method, which is *directly aimed at prediction*. It is nonimputational, i.e. we do not make guesses as to the missing values in X.

Note carefully:

- In describing our methods as being for regression applications, *we do NOT mean imputing missing values through some regression technique*; again, our technique is non-imputational. Instead, our context is that of regression applications themselves, with the goal being direct prediction of Y.
- The term *regression function* does not necessarily imply a linear model. It could also be, say, a logistic model, random forests, etc.

Illustration via the vocabulary data example

Consider the above illustration, in which we wish to predict a new case in which gender and mother's educational level are missed. Then our prediction might be the estimated value of the regression function of wage on age, birth order and ethnicity, i.e. the *marginal regression function* of wage on that set of variables.

Since each new case to be predicted will likely have a different pattern of which variables are missing, we would need to estimate many (potentially 32) marginal regression functions. For datasets with p predictors, 2 to the power p of these would be needed. This would in many applications be computationally infeasible, as each marginal model would need to be fitted and run through diagnostic plots, hyperparameter investigation, and the like.

But the Tower Property provides an alternative. It tells us that *we can obtain all the marginal regression functions from the full one.*

The Tower Property

There is a theorem in abstract probability theory that says for random variables Y, U and V,

```
E[ E(Y|U,V) | U ] = E(Y | U)
```

Though abstract, it is intuitive. Say Y, U and V are Wage, Gender and Occupation. E(Y | U,V)) is the mean wage among all workers of a given gender, in a given occupation. If we average that quantity over men and women, but still keep occupation fixed, we obtain the mean wage in that occupation.

In terms of regression functions, this says that if we take the regression function of Y on U and V, and average it over V for fixed U, we get the regression function of Y on U. If V is missing but U is known, this is very useful, as we will now explain.

How it solves our problem

In our vocabulary example above, for a new case in which age, birth order and ethnicity are known but for whom gender and mother's education are missing, we would have

U = (age, birth order, ethnicity) V = (gender,mom_ed)

E(Y|U) is the target marginal regression function that we wish to estimate and then use to predict the new case in hand. The Tower Property implies that we can obtain that estimate by the averaging process described above.

Specifically, we fit the full model to the complete cases in the data, then average that model over all data points whose values for which the values of age, birth order and ethnicity match those in the new case to be predicted. *Thus only the full model need be estimated, rather than 2 to the power p models.*

Our package *toweranNA* ("tower analysis with NAs") takes this approach. Usually, there may not be many data points having the exact value specified for U, if any, so we average over a neighborhood of points near that value. The argument *k* specifies the number of near neighbors. Since we are averaging fitted regression estimates, which are by definition already smoothed, a small value of *k* should work well.

Moreover, an early *Biometrika* paper by one of us (summarized in (Matloff, 2017, Sec. 7.5)) proved that regression averaging improves estimation of means, even with no MVs, thus an added bonus.

Classification Applications

Say we wish to predict whether the child has a vocabulary of more than 100 words. Dichotomous Y in the package must be an R factor, with the argument *yesYVal* specifying which level of the factor we wish to be considered the positive case.

```
engl2 <- engl1
engl2$vocab <- as.factor(engl2$vocab > 100)
newx
#   age birth_order ethnicity sex mom_ed
# 5  24      Second     White  NA     NA
towerOut <- makeTower(engl2,'vocab','glm',yesYVal='TRUE')
predict(towerOut,newx)
# 0.9833677 98
```

Let's try predicting birth order.

```
newx <- engl1[5,-2]
newx$age <- 28
newx$mom_ed <- NA
```

```
newx
#   age ethnicity    sex mom_ed vocab
# 5  28     White Female     NA   480
towerOut <- makeTower(engl1,'birth_order','towerKNN',opts=list(kmax=25))
predict(towerOut,newx,10)
#     Eighth Fifth First Fourth Second Seventh Sixth Third
#          0 0.004 0.568      0   0.34       0     0 0.088
```

Application to Time Series

One can handle missing values in a time series, by converting to a data frame, then applyng Tower.

Example: Gold time series

Rob Hyndman's *forecast* package includes a time series *gold*, consisting of 1108 daily gold prices. The series does have some NAs, including two in the final 10 data points:

```
data(gold,package='forecast')
gold[1099:1108]
# [1] 395.30 394.10 393.40 396.00     NA     NA 391.25 383.30 384.00 382.30
```

Let's predict the 1109th data point:

```
towerTS(gold,5,1)  # lag 5, k = 1, 'lm' etc.
# 385.2088
```

Internallly, the function *regtools::TStoX()* transforms the data to an 6-column matrix, designed for analysis of lag 5. Column 6 then becomes Y, with columns 1:5 being X. So, the call to *lm()* is loosely autoregressive, with each time point predicted from the previous 5.

Could Other MV Packages Do Prediction?

Could a *predict()* method be added to packages like `Amelia` and `mice`?

The answer on one level is no. The multiple imputation (MI) algorithms they use are designed solely to "fill in the blanks" in the training data. There is nothing in the algorithms on dealing with MVs in new cases.

On the other hand, some remedies are possible:

- Each time we are presented with a new case having MVs, we could add it to the original training data, with Y also treated as an MV. We could then rerun the MI algorithm, and the filled-in spot for the new Y would be our predicted value. (With multiple imputations, we could, say, take the mean of the filled-in Ys. Of course, this would have the drawback of entailing large increases in computation.
- Each time we are presented with a new case having MVs, we could find the nearest filled-in row in the training set, and take its Y value as our prediction for the new case. (Or look at k near neighbors and average their Ys.) As we do here with Tower, distances would be calculated on the basis of the intact values in the new case.

Thus Amelia, mice etc. could be "Tower-ized". They would not have the central focus on Y that Tower has, but reasonable *predict()* methods could be developed for them via Tower-ization.

Intuition might suggest that such an approach may be less accurate in predicting Y, as the MI algorithms are in essence devoting the data resources to predicting all columns simultaneously. This would make an interesting avenue for research, and we will be adding Tower-ization methods for Amelia etc.

Also, what about regression functions and packages that do focus on predicting Y and do allow missing X values in the training data? Random forests packages such as *randomForest* and *grf* come to mind. But again, these tolerate MVs at the training stage but not in new cases.

Assumptions

Compared to most MV packages, *toweranNA* has far less restrictive assumptions. E.g. Amelia assumes multivariate normality of the X vector, an assumption not even approximately met when some components of X are categorical variables. The mice package has extensive features for handling such cases, but there are still attendant assumptions involved.

Both of those packages, and most others, make the standard Missing at Random (MAR) assumption. What about Tower?

In our Tower Method, the assumption involves Y:

E(Y | U, V <- NA) = E(Y | U)

where VNA is a boolean variable symbolizing that the variables in V are missing.

This assumption is neither implies nor is implied by MAR, but it is similar to that condition. As with MAR, this assumption is not verifiable, but in prediction applicants, the assumption are not so vital. We simply ask, "Does it work?", meaning how well does it predict new cases? And *that* is verifiable, via cross-validation.

Reference:

Statistical Regression and Classification: from Linear Models to Machine Learning, N. Matloff, Chapman and Hall, 2017.

---

english *English vocabulary data*

Description

The Stanford WordBank data on vocabulary acquisition in young children. The file consists of about 5500 rows. There are many NA values, though, and only about 2800 complete cases. The main variables are age, birth order, sex, mother's education and vocabulary size.

makeTower *Nonimputational method for dealing with NA values in prediction application*

Description

In a prediction application, the intended regression model is applied to complete cases, from which marginal regression models can be derived for predicting new cases having arbitrary NA patterns.

Usage

```
makeTower(data,yName,regFtnName,opts,scaling=NULL,yesYVal=NULL)
## S3 method for class 'tower'
predict(object,newx,k=1,...)
```

Arguments

data	Data frame or equivalent.
yName	Name of the column in data to be predicted.
regFtnName	Regression model to be used, currently 'lm', 'glm' (family=binomial), or 'towerKNN'.
opts	Optional arguments for regFtnName, an R list.
k	number of nearest neighbors
scaling	Scaling to be applied to x and newx. Default NULL means no scaling.
yesYVal	In the case of dichotomous Y, this specifies the level to be considered positive, i.e. for which Y will be 1.
object	Object of type 'tower'.
newx	New case(s) to be predicted, in the same format as in the non-Y portion of data.
...	Other arguments need by regFtnName.

Value

Object of class 'tower', to be used as input to predict.tower.

Author(s)

Norm Matloff, Pete Mohanty

Examples

```
towerOut <- makeTower(mtcars,'mpg','lm')

newx <- mtcars[-c(1:10),-1]
for(i in 1:10)
  newx[i, i] <- NA
```

```
head(newx)
#                     cyl  disp  hp drat    wt  qsec vs am gear carb
# Merc 280C            NA 167.6 123 3.92 3.440 18.90  1  0    4    4
# Merc 450SE            8    NA 180 3.07 4.070 17.40  0  0    3    3
# Merc 450SL            8 275.8  NA 3.07 3.730 17.60  0  0    3    3
# Merc 450SLC           8 275.8 180   NA 3.780 18.00  0  0    3    3
# Cadillac Fleetwood    8 472.0 205 2.93    NA 17.98  0  0    3    4
# Lincoln Continental   8 460.0 215 3.00 5.424    NA  0  0    3    4

predict(towerOut,newx,k=3)
#  [1] 20.00086 15.17132 15.17132 15.17132 11.15469 11.15469 11.15469 28.52625
#  [9] 29.06067 28.52625 24.72144 17.45622 16.75827 15.52077 14.95958 28.52625
# [17] 25.34890 26.08506 15.52077 19.19484 15.37239 24.72144
```

`towerTS` *Tower for Times Series*

Description

Fits a linear model or other regression method to to lagged elements, using the Tower approach. Here k is the number of nearest neighbors as in `toweranNA`. Currently predicts only the component that is just past the end of the data.

Usage

```
towerTS(xts,lag,k,regFtnName='lm')
```

Arguments

`xts`	A time series.
`lag`	Lag. A positive integer.
`regFtnName`	Regression model to be used, currently 'lm', 'glm' (`family=binomial`), or 'towerKNN'.
`k`	k nearest neighbors

Details

See ?towerLM and ?regtools::TStoX

Value

Predicted value for the next item in the series.

Author(s)

Norm Matloff, Pete Mohanty

Examples

```
# create noisy cylcic series
set.seed(2020)
x <- rnorm(1000)
x <- runif(1000) * sin(x) + rnorm(1000)
w <- x
# introduce 10 percent missingness
x[sample(1000, 100)] <- NA
# make predictions with lag 3 using k=4 nearest neighbors
towerTS(x, lag=10, k=4)
#  -0.1685019
```

Index