

# Package ‘polyreg’

November 22, 2021

**Title** Polynomial Regression

**Version** 0.7.0

**Maintainer** Norm Matloff <matloff@cs.ucdavis.edu>

**Description** Automate formation and evaluation of polynomial regression models. The motivation for this package is described in 'Polynomial Regression As an Alternative to Neural Nets' by Xi Cheng, Bohdan Khomtchouk, Norman Matloff, and Pete Mohanty (<[arXiv:1806.06850](https://arxiv.org/abs/1806.06850)>).

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.5)

**Suggests** rmarkdown, knitr

**VignetteBuilder** knitr

**URL** <https://github.com/matloff/polyreg>

**BugReports** <https://github.com/matloff/polyreg/issues>

**RoxygenNote** 6.0.1

**Imports** nnet, dummies, parallel, stats, utils

**NeedsCompilation** no

**Author** Norm Matloff [aut, cre] (<<https://orcid.org/0000-0001-9179-6785>>),  
Xi Cheng [aut],  
Pete Mohanty [aut] (<<https://orcid.org/0000-0001-8531-3345>>),  
Bohdan Khomtchouk [aut],  
Matthew Kotila [aut],  
Robin Yancey [aut],  
Robert Tucker [aut],  
Allan Zhao [aut],  
Tiffany Jiang [aut]

**Repository** CRAN

**Date/Publication** 2021-11-22 05:30:02 UTC

## R topics documented:

FSR	2
getPoly	4
misc	6
pef	6
polyFit	7
predict.FSR	9
weatherTS	9

<b>Index</b>	<b>10</b>
--------------	-----------

---

FSR	<i>FSR</i>
-----	------------

---

### Description

FSR

### Usage

```
FSR(Xy, max_poly_degree = 3, max_interaction_degree = 2,
    outcome = NULL, linear_estimation = FALSE,
    threshold_include = 0.01, threshold_estimate = 0.001,
    min_models = NULL, max_fails = 2, standardize = FALSE,
    pTraining = 0.8, file_name = NULL, store_fit = "none",
    max_block = 250, noisy = TRUE, seed = NULL)
## S3 method for class 'FSR'
summary(object, estimation_overview = TRUE,
    results_overview = TRUE, model_number = NULL, ...)
## S3 method for class 'FSR'
print(x, ...)
```

### Arguments

Xy	matrix or data.frame; outcome must be in final column. Categorical variables (> 2 levels) should be passed as factors, not dummy variables or integers, to ensure the polynomial matrix is constructed properly.
max_poly_degree	highest power to raise continuous features; default 3 (cubic).
max_interaction_degree	highest interaction order; default 2 (allow $x_i * x_j$ ). Also interacts each level of factors with continuous features.
outcome	Treat y as either 'continuous', 'binary', 'multinomial', or NULL (auto-detect based on response).

<code>linear_estimation</code>	Logical: model outcome as linear and estimate with ordinary least squares? Recommended for speed on large datasets even if outcome is categorical. (For multinomial outcome, this means treated response as vector.) If FALSE, estimator chosen based on 'outcome' (i.e., OLS for continuous outcomes, <code>glm()</code> to estimate logistic regression models for 'binary' outcomes, and <code>nnet::multinom()</code> for 'multinomial').
<code>threshold_include</code>	minimum improvement to include a recently added term in the model (change in fit originally on 0 to 1 scale). -1.001 means 'include all'. Default: 0.01. (Adjust $R^2$ for linear models, Pseudo $R^2$ for logistic regression, out-of-sample accuracy for multinomial models. In latter two cases, the same adjustment for number of predictors is applied as pseudo- $R^2$ .)
<code>threshold_estimate</code>	minimum improvement to keep estimating (pseudo $R^2$ so scale 0 to 1). -1.001 means 'estimate all'. Default: 0.001.
<code>min_models</code>	minimum number of models to estimate. Defaults to the number of features (unless $P > N$ ).
<code>max_fails</code>	maximum number of models to <code>FSR()</code> can fail on computationally before exiting. Default == 2.
<code>standardize</code>	if TRUE (not default), standardizes continuous variables.
<code>pTraining</code>	portion of data for training
<code>file_name</code>	If a file name (and path) is provided, saves output after each model is estimated as an .RData file. ex: <code>file_name = "results.RData"</code> . See also <code>store_fit</code> for options as to how much to store in the outputted object.
<code>store_fit</code>	If <code>file_name</code> is provided, <code>FSR()</code> will return coefficients, measures of fit, and call details. Save entire fit objects? Options include "none" (default, just save those other items), "accepted_only" (only models that meet the threshold), and "all".
<code>max_block</code>	Most of the linear algebra is done recursively in blocks to ease memory management. Default 250. Changing up or down may slow things...
<code>noisy</code>	display measures of fit, progress, etc. Recommended.
<code>seed</code>	Automatically set but can also be passed as parameter.
<code>estimation_overview</code>	logical: describe how many models were planned, sample size, etc.?
<code>results_overview</code>	logical: give overview of best fit model, etc?
<code>model_number</code>	If non-null, an integer indicating which model to display a summary of.
<code>object</code>	an FSR object, can be used with <code>predict()</code> .
<code>x</code>	an FSR object, can be used with <code>print()</code> .
<code>...</code>	ignore.

**Value**

list with slope coefficients, model and estimation details, and measures of fit (object of class 'FSR').

**Examples**

```
out <- FSR(mtcars)
```

---

<code>getPoly</code>	<i>Get polynomial terms</i>
----------------------	-----------------------------

---

**Description**

Generate polynomial terms of predictor variables for a data frame or data matrix.

**Usage**

```
getPoly(xdata = NULL, deg = 1, maxInteractDeg = deg,
        Xy = NULL, standardize = FALSE,
        noisy = TRUE, intercept = FALSE, returnDF = TRUE,
        modelFormula = NULL, retainedNames = NULL, ...)
```

**Arguments**

<code>xdata</code>	Data matrix or data frame without response variable. Categorical variables (> 2 levels) should be passed as factors, not dummy variables or integers, to ensure the polynomial matrix is constructed properly.
<code>deg</code>	The max degree of power terms. Default 1 so just returns model matrix by default.
<code>maxInteractDeg</code>	The max degree of nondummy interaction terms. $x_1 * x_2$ is degree 2. $x_1^3 * x_2^2$ is degree 5. Implicitly constrained by <code>deg</code> . For example, if <code>deg = 3</code> and <code>maxInteractDegree = 2</code> , $x_1^1 * x_2^2$ (i.e., degree 3) will be included but $x_1^2 * x_2^2$ (i.e., degree 4) will not.
<code>Xy</code>	The dataframe with the response in the final column (provide <code>xdata</code> or <code>Xy</code> but not both). Categorical variables (> 2 levels) should be passed as factors, not dummy variables or integers, to ensure the polynomial matrix is constructed properly.
<code>standardize</code>	Standardize all continuous variables? (Default: FALSE.)
<code>noisy</code>	Output progress updates? (Default: TRUE.)
<code>intercept</code>	Include intercept? (Default: FALSE.)
<code>returnDF</code>	Return a data.frame (as opposed to model.matrix)? (Default: TRUE.)
<code>modelFormula</code>	Internal use. Formula used to generate the training model matrix. Note: anticipates that polynomial terms are generated using internal functions of <code>library(polyreg)</code> . Also, providing <code>modelFormula</code> bypasses <code>deg</code> and <code>maxInteractDeg</code> .
<code>retainedNames</code>	Internal use. colnames of <code>polyMatrix</code> object <code>\$xdata</code> . Requires <code>modelFormula</code> be inputted as well.
<code>...</code>	Additional arguments to be passed to <code>model.matrix()</code> via <code>polyreg:::model_matrix()</code> . Note <code>na.action = "na.omit"</code> .

## Details

The `getPoly` function takes in a data frame or data matrix and generates polynomial terms of predictor variables.

Note the subtleties involving dummy variables. The square, cubic and so on terms are the same as the original variable, and the various duplicates must be eliminated.

Similarly, after dummy variable are created from a categorical variable having more than two levels, the resulting columns will be orthogonal to each other. In almost all cases, this argument should be set to `TRUE` at the training stage, and then in predictions one should use the vector of names in the component in the return value; `predict.polyFit` does the latter automatically.

Note: If a column that is an R factor has levels with spaces in the names, this will interfere with the parsing, and must be avoided.

## Value

The return value of `getPoly` is a `polyMatrix` object. This is an S3 class containing a `model.matrix` `xdata` of the generated polynomial terms. The predictor variables have column names `V1`, `V2`, etc. The object also contains `modelFormula`, the formula used to construct the model matrix, and `XtestFormula`, the formula which should be used out-of-sample (when `y_test` is not available).

## Examples

```
N <- 125
rawdata <- data.frame(x1 = rnorm(N),
                     x2 = rnorm(N),
                     group = sample(letters[1:5], N, replace=TRUE),
                     z = sample(c("treatment", "control"), N, replace=TRUE),
                     result = sample(c("win", "lose", "tie"), N, replace=TRUE))

head(rawdata)

P <- length(levels(rawdata$group)) - 1 +
      length(levels(rawdata$z)) - 1 +
      length(levels(rawdata$result)) - 1 +
      sum(unlist(lapply(rawdata, is.numeric)))

# quadratic polynomial, includes interactions
# since maxInteractDeg defaults to deg
X <- getPoly(rawdata, 2)$xdata
ncol(X) # 40

# cubic polynomial, no interactions
X <- getPoly(rawdata, 3, 1)$xdata
ncol(X) # 13

# cubic polynomial, interactions
X <- getPoly(rawdata, 3, 2)$xdata
ncol(X) # 58

# cubic polynomial, interactions
X <- getPoly(rawdata, 3)$xdata
ncol(X) # 101
```

```

# making final column the response variable, y
# results in TRUE (fewer columns)
ncol(getPoly(Xy=rawdata, deg=2)$xdata) < ncol(getPoly(rawdata, 2)$xdata)

# preparing polynomial matrices for crossvalidation
# getPoly() returns a polyMatrix() object containing XtestFormula
# which should be used to ensure factors are handled correctly out-of-sample
Xtrain <- getPoly(rawdata[1:100,],2)
Xtest <- getPoly(rawdata[101:125,], 2, modelFormula = Xtrain$XtestFormula)

```

---

misc

*Miscellaneous*


---

## Description

Utilities

## Usage

```
toFactors(df, cols)
```

## Arguments

df	A data frame.
cols	A vector of column numbers.

## Details

The `toFactors` function converts each `df` column in `cols` to a factor, returns new version of `df`. Should be used on categorical variables stored as integer codes before calling the library's main functions, including `getPoly`, `FSR`, or `polyFit`.

---

pef

*Silicon Valley programmers and engineers data*


---

## Description

This data set is adapted from the 2000 Census (5% sample, person records). It is mainly restricted to programmers and engineers in the Silicon Valley area. (Apparently due to errors, there are some from other ZIP codes.)

Has columns for age, education, occupation, gender, wage income and weeks worked. The education column has been collapsed to Master's degree, PhD and other.

The variable codes, e.g. occupational codes, are available from the Census Bureau, at <https://www.census.gov/prod/cen2000/doc/pums.pdf>. (Short code lists are given in the record layout, but longer ones are in the appendix Code Lists.)

**Usage**

```
data(pef)
```

---

polyFit

*Polynomial Fit*

---

**Description**

Fit polynomial regression using a linear or logistic model; predict new data.

**Usage**

```
polyFit(xy, deg, maxInteractDeg=deg, use = "lm", glmMethod="one",
        return_xy=FALSE, returnPoly=FALSE, noisy=TRUE)
## S3 method for class 'polyFit'
predict(object, newdata, ...)
```

**Arguments**

xy	Data frame with response variable in the last column. Latter is numeric, except in the classification case: Categorical variables (> 2 levels) must be passed as factors or character variables if use is 'glm'; an integer vector must be used for the for the 'mvrlm' case, or for the 2-class case (0s and 1s).
deg	The max degree for polynomial terms. A term such as uv, for instance, is considered degree 2.
maxInteractDeg	The max degree of interaction terms.
use	Set to 'lm' for linear regression, 'glm' for logistic regression, or 'mvrlm' for multivariate-response lm.
glmMethod	Defaults to 'one,' meaning the One Versus All Method. Use 'all' for All Versus All.
newdata	Data frame, one row for each "X" to be predicted. Must have the same column names as in xy (without "Y").
object	An item of class 'polyFit' containing output. Can be used with predict().
return_xy	Return data? Default: FALSE
returnPoly	return polyMatrix object? Defaults to FALSE since may be quite large.
noisy	Logical: display messages?
...	Additional arguments for getPoly().

## Details

The `polyFit` function calls `getPoly` to generate polynomial terms from predictor variables, then fits the generated data to a linear or logistic regression model. (Powers of dummy variables will not be generated, other than degree 1, but interaction terms will be calculated.)

When logistic regression for classification is indicated, with more than two classes, All-vs-All or One-vs-All methods, coded 'all' and 'one', can be applied to deal with multiclass problems.

Under the 'mvrlm' option in a classification problem, `lm` is called with multivariate response, using `cbind` and dummy variables for class membership as the response. Since predictors are used to form polynomials, this should be a reasonable model, and is much faster than 'glm'.

## Value

The return value of `polyFit()` is a `polyFit` object. The original arguments are retained, along with the fitted models and so on.

The prediction function `predict.polyFit` returns the predicted value(s) for new data. It also contains probability for each class as an attribute named `prob`. In the classification case, these will be the predicted class labels, 1,2,3,...

## Examples

```
N <- 125
xyTrain <- data.frame(x1 = rnorm(N),
                      x2 = rnorm(N),
                      group = sample(letters[1:5], N, replace=TRUE),
                      score = sample(100, N, replace = TRUE) # final column is y
                      )

pfOut <- polyFit(xyTrain, 2)

# 4 new test points
xTest <- data.frame(x1 = rnorm(4),
                   x2 = rnorm(4),
                   group = sample(letters[1:5], 4, replace=TRUE))

predict(pfOut, xTest) # returns vector of 4 predictions

data(pf)
# predict wageinc
z <- polyFit(pf[,c(setdiff(1:6,5),5)],2)
predict(z,pf[2000,c(setdiff(1:6,5),5)]) # 56934.39
# predict occ
z <- polyFit(pf[,c(setdiff(1:6,3),3)],2,use='glm')
predict(z,pf[2000,c(setdiff(1:6,3),3)]) # '100', probs 0.43, 0.26,...
```



---

predict.FSR	<i>predict.FSR</i>
-------------	--------------------

---

**Description**

predict.FSR

**Usage**

```
## S3 method for class 'FSR'
predict(object, newdata, model_to_use = NULL,
        standardize = NULL, noisy = TRUE, ...)
```

**Arguments**

object	FSR output. Predictions will be made based on object\$best_formula unless model_to_use is provided (as an integer).
newdata	New Xdata.
model_to_use	Integer optionally indicating a model to use if object\$best_formula is not selected. Example: model_to_use = 3 will use object\$models\$formula[3].
standardize	Logical—standardize numeric variables? (If NULL, the default, bypasses and decides based on object\$standardize.)
noisy	Display output?
...	ignore

**Value**

y\_hat (predictions using chosen model estimates).

**Examples**

```
out <- FSR(mtcars[1:30,])
forecast <- predict(out, mtcars[31:nrow(mtcars),])
```

---

weatherTS	<i>Weather Time Series</i>
-----------	----------------------------

---

**Description**

Various measurements on weather variables collected by NASA. Downloaded via nasapower; see that package for documentation.

# Index

FSR, [2](#)

getPoly, [4](#)

misc, [6](#)

pef, [6](#)

polyFit, [7](#)

polyMatrix (getPoly), [4](#)

predict.FSR, [9](#)

predict.polyFit (polyFit), [7](#)

print.FSR (FSR), [2](#)

summary.FSR (FSR), [2](#)

toFactors (misc), [6](#)

weatherTS, [9](#)