The LinTester Class

November 17, 2012

Brief Overview

The LinTester class was written for Harish Bhat's Math 32 class. It is designed to facilitate the organization and rapid testing of linear regression models for larger datasets. The overall function is very simple to use, but as of this time of writing, it has not been extensively tested. Expect lots of bugs.

Usage

To instantiate an instance of LinTester, use the function lintester(). Lintester requires three arguments: the data to use for predictions (predictors or covariates) , the data to be predicted (response variable), and a formula to use to form the model. The predictors must be in a data frame, the response must be a vector, and the formula must be a formula which does *not* reference the parent dataframe (see Paragraph 2 of "More Details" for details).

To take an example, let us load a data frame titled "carstats" with the columns "tire," "gas," and "speed." To construct the LinTester object with a model that has speed as a function of tire and gas, we would do:

```
load('carstats.RData')
myFormula = formula(speed ~ tire + gas)
myLinTester = lintester(carstats, carstats\$speed, myFormula)
```

myLinTester would now contain an object that is ready to model speed with respect to tire and gas.

When lintester() is called, an additional column of data is appended on to the dataframe. This column contains the "bin number" for n-fold cross validation purposes. These numbers will stay fixed for as long as the object is in existence. The advantage of this method is that multiple models can be tried on the exact same bins, and if certain bins appear to be outliers, they can easily be examined with a which() command.

What if we want to rescramble the bins, for example, when testing one model many times? To do this, we use the LinTester's rebin() method. Calling rebin() will cause LinTester to scramble the bins and reassign bin numbers.

```
myLinTester = rebin(myLinTester)
```

In addition to this, LinTester has accessor and mutator functions. To access the "data" dataframe in LinTester, use getData(myLinTester). Similar accessors getResponse(), getForm(), and getErrors() return the response vector, the formula, and the error matrix, respectively.

The mutator functions allow the user to change the values stored in the LinTester. To use a mutator function, call it with the object and the data to be set, e.g. setData(myLinTester, newdataframe). setResponse() and setForm() can be used to set the response vector and formula. The setter functions, however, are error-checked against length (same as the constructor). If you want to set response and data of a different length, use the setAll() function. setAll() sets both data and response simultaneously.

More Details

The LinTester is a class, meaning that it has certain defined methods and data values. Any instance of a LinTester object must have at least input data (which must be of class

```
data.frame
```

) and a response variable (of class

```
vector
```

. The class constructor will allow the object to be built unless the data and response are both non-zero, and have the same number of entries. In addition, the LinTester has a formula, which is used to make the regression, and an "errors" dataframe, which displays the results from n-fold cross validation regression testing.

It is crucial that the formula NOT contain any references to its parent data, and use only column names. For example, if the predictor dataframe is "cancer," with columns "smoker" and "lung_cancer", an appropriate formula is

```
formula(lung_cancer ~ smoker)
```

.

Attempting to use

```
formula(cancer\$lung_cancer ~ cancer\$smoker)
```

will cause the predictions to fail.

One can also easily change the number of bins used to cross validation. To do this, call rebin(myLinTester, bin_number), where bin_number is a positive integer.