WhatIf: Software for Evaluating Counterfactuals[1]

Heather Stoll[2] Gary King[3] Langche Zeng[4]

Version 1.5-5
August 12, 2010

[1]Available from http://GKing.Harvard.Edu/whatif.

[2]Assistant Professor of Political Science, University of California, Santa Barbara (3713 Ellison Hall, University of California, Santa Barbara, CA 93106; http://www.polsci.ucsb.edu/faculty/hstoll/, hstoll@polsci.ucsb.edu).

[3]David Florence Professor of Government, Harvard University (Institute for Quantitative Social Science, 34 Kirkland Street, Harvard University, Cambridge MA 02138; http://GKing.Harvard.Edu, King@Harvard.Edu, (617) 495-2027).

[4]Professor of Political Science, University of California-San Diego (Department of Political Science, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0521, zeng@ucsd.edu).

Contents

1 Introduction

WhatIf implements the methods for evaluating counterfactuals introduced in King and Zeng (2006) and King and Zeng (2007):

> Gary King and Langche Zeng. 2006. "The Dangers of Extreme Counterfactuals," *Political Analysis* 14 (2): 131–159.

and

> Gary King and Langche Zeng. 2007. "When Can History Be Our Guide? The Pitfalls of Counterfactual Inference," *International Studies Quarterly* 51 (March): 183–210.

The two papers overlap, with the first containing all the proofs and technical material and the second having more pedagogical material and examples.

Inferences about counterfactuals are essential for prediction, answering "what if" questions, and estimating causal effects. However, when the counterfactuals posed are too far from the data at hand, conclusions drawn from well-specified statistical analyses become based largely on speculation hidden in convenient modeling assumptions that few would be willing to defend. Unfortunately, standard statistical approaches assume the veracity of the model rather than revealing the degree of model dependence, which makes this problem hard to detect.

WhatIf offers easy-to-apply methods to evaluate counterfactuals that do not require sensitivity testing over specified classes of models. If an analysis fails the tests offered here, then we know that substantive inferences will be sensitive to at least some modeling choices that are not based on empirical evidence, no matter what method of inference one chooses to use. Specifically, WhatIf will indicate whether a given counterfactual is an extrapolation (and therefore risking more model dependence) or a (safer) interpolation. Using an algorithm developed in King and Zeng (2006) to identify whether counterfactual points are within the convex hull of the observed data, this is feasible even for large numbers of explanatory variables. It will also compute either the Gower or Euclidian distances from the counterfactuals to each observed data point. The convex hull test can additionally be used to approximate the common support of the treatment and control groups in causal inference. Numerical and graphic summaries are offered.

WhatIf has been incorporated in MatchIt, and also works easily with Zelig output (Ho et al., 2007; Imai, King and Lau, 2006, 2008).

2 Installation

WhatIf requires `R` version 2.3.1 or later, available from CRAN (`http://cran.r-project.org/`), and the package `lpSolve`, also available from CRAN. Installation of WhatIf differs slightly by operating system.

2.1 Windows

Begin the installation process by launching `R`. To install the package WhatIf as well as the package that it depends upon, `lpSolve`, type:

```
> install.packages("WhatIf", dependencies = TRUE)
```

at the `R` command prompt. This command installs the packages from the CRAN respository set as part of your `R` options. You can see what its current value is by calling

```
> getOption("repos")
```

The default, 'factory fresh' setting will usually prompt you to select a CRAN mirror. One alternative for installing the package WhatIf is to use Gary King's website as the repository. You can do this by typing:

```
> install.packages("WhatIf", repos = "http://gking.harvard.edu")
```

You will then need to install the package `lpSolve` from CRAN by typing:

```
> install.packages("lpSolve")
```

A second alternative is to download the Windows bundle from `http://GKing.Harvard.Edu/bin/windows/contrib` and use the `R` pull-down menu commands for installing a package from a zip file. You again will then need to install `lpSolve`, which can be done either by typing the command given above at the command prompt or by using the pull-down menus. Finally, you then only need to type:

```
> library("WhatIf")
```

from within `R` to load the WhatIf package, after which you may begin working with it.

2.2 Linux/Unix

You initially need to create both local `R` and local `R` library directories if they do not already exist. At the Linux/Unix command prompt in your home directory, do this by typing:

```
> mkdir ~/.R ~/.R/library
```

Then open the '`.Renviron`' file that resides in your home directory, creating it if necessary, and add the line:

```
R_LIBS = "~/.R/library"
```

using your preferred text editor. These steps only need to be performed once. To install the package WhatIf as well as the package that it depends upon, `lpSolve`, type:

```
> install.packages("WhatIf", dependencies = TRUE)
```

at the `R` command prompt. This command installs the packages from the CRAN respository set as part of your `R` options. You can see what its current value is by calling

```
> getOption("repos")
```

The default, 'factory fresh' setting will usually prompt you to select a CRAN mirror. One alternative for installing the package WhatIf is to use Gary King's website as the repository. You can do this by typing:

```
> install.packages("WhatIf", repos = "http://gking.harvard.edu")
```

A second alternative is to download the Linux/Unix bundle '`WhatIf_XX.tar.gz`', available from `http://gking.harvard.edu/R/CRAN/src/contrib/`, and place it in your home directory. Note that '`XX`' is the current version number. Then, at the Linux/Unix command line from your home directory, type

```
> R CMD INSTALL WhatIf_XX.tar.gz
```

to install the package. Finally, you then only need to type:

```
> library("WhatIf")
```

from within `R` to load the WhatIf package, after which you may begin working with it.

3 Examples

3.1 Counterfactuals about U.N. Peacekeeping

This section illustrates the workings of WhatIf with the empirical example in Section 2.4 of King and Zeng (2006), which evaluates counterfactuals about the causal impact of U.N. peacekeeping operations on peacebuilding success.

The factual data set has 124 observations (including two with missing values) on ten covariates as well as on the key causal variable, `untype4`, which is a dummy variable. The counterfactual data set is the observed covariate data set with `untype4` replaced with 1−`untype4`. We list-wise delete the two counterfactuals that are not fully observed. We then save the two data sets, one factual and the other counterfactual, as text files in our current working directory and name them '`peacef.txt`' and '`peacecf.txt`', respectively. The first five rows of '`peacef.txt`' look like:

```
  decade wartype   logcost wardur factnum factnumsq    trnsfcap untype4 treaty
1      5       1 14.917450     72       4        16    5.735545       0      0
2      4       0 15.671810    168       6        36    9.730863       0      0
3      5       1  6.907755     24       2         4   12.626030       0      0
4      5       1 12.971540     24       2         4 -112.000000       0      1
5      3       1  9.210340    216       2         4    4.275317       0      0
    develop       exp
1  132.8466 0.1217277
2  132.0000 0.1163292
3 1533.0000 0.0610000
4 2216.6080 0.1294513
5 1295.0000 0.1420000
```

Similarly, the first five rows of '`peacecf.txt`' look like:

```
  decade wartype   logcost wardur factnum factnumsq    trnsfcap 1-untype4
1      5       1 14.917450     72       4        16    5.735545         1
2      4       0 15.671810    168       6        36    9.730863         1
3      5       1  6.907755     24       2         4   12.626030         1
4      5       1 12.971540     24       2         4 -112.000000         1
5      3       1  9.210340    216       2         4    4.275317         1
  treaty   develop       exp
1      0  132.8466 0.1217277
2      0  132.0000 0.1163292
3      0 1533.0000 0.0610000
4      1 2216.6080 0.1294513
5      0 1295.0000 0.1420000
```

The function `whatif` can be called in two alternative ways to analyze these counterfactuals. First, typing:

```
> my.result <- whatif(data = "peacef.txt", cfact = "peacecf.txt")
```

tells `whatif` to load the datasets '`peacef.txt`' and '`peacecf.txt`' from our working directory. Second, typing:

```
> my.result <- whatif(data = peacef, cfact = peacecf)
```

tells `whatif` to use the `R` objects `peacef` and `peacecf` loaded into memory prior to the function call. These objects must be either non-character matrices or data frames containing the counterfactual and observed covariate data, respectively; in this case, they are data frames. Alternatively, `peacef` may be either a Zelig or other `R` model output object (e.g., a model output object returned by a call to `glm`).

The resulting output object `my.result` is a five-element list (six-element if the option "return.distance=T" is used), each element of which we now describe. The first is simply the call. The second is a logical vector named `in.hull`, which contains the results of the convex hull test. Each element can have a value of either `FALSE`, indicating that the corresponding counterfactual is not in the convex hull of the observed data and thus requires extrapolation, or `TRUE`, indicating the opposite. To see the values of `in.hull`, we type:

```
> my.result$in.hull
```

For this example, the values are all `FALSE`.

The third element of the output list, `geom.var`, is the geometric variability of the observed data, which we retrieve by typing:

```
> my.result$geom.var
```

In this case, it is 0.110 when rounding to three significant digits. King and Zeng offer the geometric variability as a rule of thumb threshold: counterfactuals with distances to the observed covariate data less than this value are to some extent nearby the data. By default, pairwise Gower's distances (G^2) between each counterfactual and data point are calculated by `whatif` in order to determine which counterfactuals are nearby the data; alternatively, `whatif` will calculate the pairwise (squared) Euclidian distance between each counterfactual and data point by setting the parameter `distance` equal to `"euclidian"` as follows:

```
> my.result <- whatif(data = peacef, cfact = peacecf, distance = "euclidian")
```

However, this option is only appropriate for quantitative data; since some of our variables are qualitative, we use the default Gower's distance measure.

Note that the matrix containing these distances can be large in size and is not returned by default. To return the distance matrix, set the parameter `return.distance` to `TRUE`.

The fourth element of the output object, `sum.stat`, is a numeric vector, each element of which is the proportion of data points nearby the corresponding counterfactual. The values can be seen by typing:

```
> my.result$sum.stat
```

The output looks like:

```
          1           2           3           4           5           6
0.008196721 0.008196721 0.008196721 0.008196721 0.008196721 0.008196721
          7           8           9          10          11          12
0.008196721 0.008196721 0.008196721 0.008196721 0.008196721 0.008196721
...
        121         122
0.008196721 0.016393443
```

The numerical summary reported on page 14 of King and Zeng (2006) is the average of `sum.stat` over all counterfactuals, which we can obtain using the command

```
> mean(my.result$sum.stat)
```

In this case, the average is 1.3 percent. This statistic is reported for your convenience by the function `summary`.

We note that by default, 'nearby' is defined as having a distance to the counterfactual less than or equal to the geometric variability of the observed data. The default can be changed by setting a value for the parameter `nearby`. For example, to instead set the nearby criterion at two geometric variances, we would type:

```
> my.result <- whatif(data = peacef, cfact = peacecf, nearby = 2)
```

The fifth element of the output object, `cum.freq`, stores information on the cumulative frequency distribution of the distances between a counterfactual and the observed covariate data. To access the cumulative frequency distribution for the default set of Gower distances (from 0 to 1 in increments of 0.5) between the first counterfactual and the data points, for example, we type:

```
> my.result$cum.freq[1, ]
```

This prints the distribution to the screen:

```
           0        0.05         0.1        0.15         0.2        0.25
0.000000000 0.000000000 0.008196721 0.081967213 0.262295082 0.483606557
         0.3        0.35         0.4        0.45         0.5        0.55
0.680327869 0.844262295 0.950819672 0.991803279 0.991803279 1.000000000
         0.6        0.65         0.7        0.75         0.8        0.85
1.000000000 1.000000000 1.000000000 1.000000000 1.000000000 1.000000000
         0.9        0.95           1
1.000000000 1.000000000
```

Alternatively, we can change the default set of Gower distances by using the parameter `freq`. For example, to calculate a cumulative frequency distribution solely for the Gower distances of 0, 0.5, and 1.0, we type:

```
> my.result <- whatif(data = peacef, cfact = peacecf, freq = c(0, 0.5, 1.0))
```

Now the cumulative frequency distribution for the first counterfactual looks as follows:

```
> my.result$cum.freq[1, ]
        0       0.5         1
0.0000000 0.9918033 1.0000000
```

We now turn to the auxiliary functions included in the WhatIf package. The first is `plot`, which produces figures that graph the cumulative frequency distribution of the distances similar to Figure 3 in King and Zeng (2006). This function takes as its input a `whatif` output object. To plot the default cumulative frequency distributions for all counterfactuals to the screen, type:

```
> plot(my.result)
```

Plotting 122 distributions on the same graph will not be very helpful, however. A particular frequency distribution or combination of frequency distributions can be plotted by setting the parameter `numcf` to equal the desired values. For example, to plot only the cumulative frequency distribution for the first counterfactual, we type:

```
> plot(my.result, numcf = 1)
```

We also have the option of smoothing the raw cumulative frequencies, which can be plotted either on their own or in addition to the raw data. The parameter controlling this option is `type`. To plot both the raw and LOWESS smoothed cumulative frequency distributions for the first two counterfactuals, for example, we type:

```
> plot(my.result, numcf = c(1, 2), type = "b")
```

where "b" stands for 'both'. Alternatively, assigning the value "l" to type would plot only the smoothed frequencies. To save the graph as an encapsulated postscript file for later use instead of printing it to the screen, we set the parameter eps equal to TRUE:

```
> plot(my.result, numcf = c(1, 2), type = "b", eps = TRUE)
```

The graph is saved to our working directory.

Not surprisingly, the function summary summarizes the most important information produced by the function whatif. The output object, a list, contains this information, which may also be printed to the screen. For example, typing:

```
> summary(my.result)
```

displays the total number of counterfactuals evaluated; the number of counterfactuals that are in the convex hull of the observed covariate data; the percentage of data points nearby each counterfactual averaged over all counterfactuals; and a table that contains both the results of the convex hull test and the percentage of data points nearby the counterfactual for each counterfactual. Alternatively, typing:

```
> my.result.sum <- summary(my.result)
```

saves the summary information as the object my.result.sum, which can be printed to the screen by typing either:

```
> print(my.result.sum)
```

or:

```
> my.result.sum
```

at the command prompt.

Finally, the package WhatIf includes two print functions. To print the output object returned by whatif to the screen, type either:

```
> print(my.result)
```

or the name of the output object at the command prompt. Not printed by these calls are the matrices of distances and cumulative frequencies. These large objects can be printed by setting the parameters print.dist (if the distance matrix was returned) and print.freq equal to TRUE, respectively. For example, to print the entire output object except for the matrix of Gower distances to the screen, we type:

```
> print(my.result, print.freq = TRUE)
```

The other print function controls the printing of the output object from the function summary.

3.2 Identifying Common Support in Causal Inference

The same algorithm for identifying whether or not counterfactuals fall within the convex hull of the observed covariate data can be used to assess common support. We illustrate here with two examples.

3.2.1 U.N. Peacekeeping

In Section 2.4 of King and Zeng (2007), the seven fully observed countries that experienced a U.N. peacekeeping mission comprise the treatment group while the remaining 115 fully observed (and 117 non-fully observed) countries that lacked a U.N. peacekeeping mission constitute the control group. Cases of the former type receive a coding of '1' on the key causal variable, `untype4`, while cases of the latter type are coded '0'.

When estimating the *average treatment effect on the treated*, we discard controls with observed covariate data not within the convex hull of the data for the treated as follows:

```
> my.result.cntrl <- whatif(formula = ~ decade + wartype + logcost +
+ wardur + factnum + factnumsq + trnsfcap + treaty + develop + exp,
+ data = peacef[peacef$untype4 == 1,], cfact = peacef[peacef$untype4 == 0,])
```

This command feeds the seven treated countries to the parameter `data` and the 117 control countries to the parameter `cfact`. This differs from the last section, where `data` contained all 124 observed data points, whether treated or not. The parameter `formula` allows us to drop `untype4` from the two data frames, which we do by naming all of the variables that we want to keep. (We eliminate this variable since our goal is to identify the convex hull of the observed pre-treatment covariates.) Note that we do not specify a dependent variable in the formula. We then look at the results:

```
> my.result.cntrl$in.hull
```

The control group countries not on the support of the treated countries are those with `FALSE` entries—in this case, all 115. Note that for this data and call, two messages designed to inform the user about choices made by `whatif` in the face of problematic data are printed to the screen. The first informs us that `whatif` has deleted the two control group cases with missing values from the `cfact` data set since counterfactuals must be fully observed. The second, "range of at least one variable equals zero", informs us that `data` contains a degenerate case: the variable `treaty` has no variance (and hence a range of zero) in the observed covariate data set of the treated countries. In order to calculate the Gower distances, `whatif` must make assumptions about the handling of such variables. Specifically, it ignores their contribution unless the values of the data point and counterfactual are identical, in which case the normalized difference is set to zero.

A different, but perhaps more reliably estimable quantity, may often be obtained by *also* dropping observations in the treatment group whose observed covariate data falls outside of the convex hull of the control group. Any countries remaining comprise the data set that lies on the common support. Both the prior and this second step can be performed simultaneously using WhatIf as originally described in Section 3.1 for evaluating counterfactuals. Accordingly, we type:

```
> peacef2cf <- peacef
> peacef2cf$untype4 <- 1 - peacef2cf$untype4
> my.result.comb <- whatif(data = peacef, cfact = peacef2cf)
```

Here, we initially create the counterfactual data set `peacef2cf` directly from the factual, replacing `untype4` with 1 - `untype4`. We could also have supplied the data set `peacecf`, originally constructed in a similar manner. (The data sets would be identical if the two counterfactuals with missing data were list-wise deleted from `peacef2cf`.) We now look at the results of the convex hull test as before and see that none of the counterfactuals are in the convex hull. Hence, there is no data on the common support of both the treatment and control groups.

3.2.2 Hypothetical Data

To demonstrate that the latter approach really does combine the individual assessments of support on the treatment and control groups, consider this hypothetical data set:

```
> sqdata <- data.frame(t = c(1, 1, 1, 1, 0, 0, 0, 0),
+                      x = c(0, 0, 1, 1, .5, .5, 1.5, 1.5),
+                      y = c(1, 0, 0, 1, .5, 1.5, .5, 1.5))
```

The variable 't' is the treatment. The convex hull of the observed covariate data of the treatment group is obviously a unit square with its lower left vertex at the origin. The convex hull of the control group is also a unit square, but one with its lower left vertex at the point $(0.5, 0.5)$ in the Cartesian plane.

We first identify the control group units that are not on the support of the treated units (i.e., the control group units not in the convex hull of the covariate data of the treated group) as follows:

```
> summary(whatif(~ x + y, data = sqdata[sqdata$t == 1,], cfact =
+ sqdata[sqdata$t == 0,]))
```

which as before uses the parameter `formula` to eliminate the treatment variable, `t`, from the data frames. Only the first unit from the control group, the point $(0.5, 0.5)$ in the Cartesian plane, is in the convex hull and hence on the support of the treated group. We next identify the treated group units that are not on the support of the control units by typing:

```
> summary(whatif(~ x + y, data = sqdata[sqdata$t == 0,], cfact =
+ sqdata[sqdata$t == 1,]))
```

The treatment group unit represented by the point $(1, 1)$ is the only one in the convex hull and hence on the support of the control group. Accordingly, if we were to eliminate the units without common support as identified by the two separate tests, we would eliminate all units save the points $(0.5, 0.5)$, the only control group unit on the support of the treated group, and $(1, 1)$, the only treated group unit on the support of the control group.

Alternatively, we can combine the two steps:

```
> summary(whatif(data = sqdata, cfact = cbind(1 - sqdata[, 1], sqdata[, 2:3])))
```

This time, two counterfactuals are in the convex hull of the data. These counterfactuals correspond to the units with observed covariate data $(0.5, 0.5)$ and $(1, 1)$. Accordingly, we conclude that only two units are on the common support, the same conclusion that we drew from the two separate tests.

3.3 Using WhatIf with Zelig

We now illustrate how WhatIf can be easily used with `Zelig`. As an example, we first generate `Zelig` output from a simple logistic model using the hypothetical data set created in the prior example:

```
> z.out <- zelig(t ~ x + y, data = sqdata, model = "logit")
```

We next create a counterfactual using the `Zelig` command `setx`:

```
> x.out <- setx(z.out, x = 2, y = 3)
```

This is normally followed by a call to the `Zelig` command `sim` to compute quantities of interest, such as predicted values given these values of the explanatory variables. See, for example, http://gking.harvard.edu/zelig/docs/Quick_Overview.html. WhatIf enables you to evaluate the values to which you set the explanatory variables before simulating quantities of interest. We do this by calling `whatif` as follows:

```
> summary(whatif(data = z.out, cfact = x.out))
```

The results indicate that this counterfactual is not in the convex hull of the data. In this situation, you may want to rethink whether or not you should proceed on to the `sim` stage of analysis. Note that if an intercept was fit as part of the original model, `whatif` automatically drops it from both the observed covariate data set extracted from the `zelig` output object `z.out` and the `setx`-generated counterfactual `x.out`.

3.4 Using WhatIf with Other R Model Output Objects

Suppose that instead of using `Zelig`, we use the function `lm` to fit a linear model to the same hypothetical data by typing:

```
> lm.out <- lm(t ~ x + y, data = sqdata)
```

In this case, we could then use WhatIf to evaluate a counterfactual as follows:

```
> summary(whatif(data = lm.out, cfact = data.frame(x = 2, y = 3)))
```

As with `zelig` output objects, intercepts are dropped from the observed covariate data sets extracted in this manner. Unlike with `Zelig`, however, counterfactuals are not generated automatically by `lm`; hence, the counterfactuals that you supply to `whatif` should not include an intercept. The parameter `formula` can be used to drop, select, and transform the variables in `data` and `cfact` when `data` is a `R` model or `zelig` output object in the same way that it can be used when `data` is a matrix or data frame. For example, to drop the variable `x`, we type:

```
> summary(whatif(~ y, data = lm.out, cfact = data.frame(x = 2, y = 3)))
```

or more simply and equivalently:

```
> summary(whatif(~ y, data = lm.out, cfact = data.frame(y = 3)))
```

If instead we decide to run the test using the square of `x`, we type:

```
> summary(whatif(~ I(x^2) + y, data = lm.out, cfact = data.frame(x = 2, y = 3)))
```

following standard `R` conventions for formulas.

3.5 Demos and Data Sets

`R` will automatically walk you through the examples related to U.N. peacekeeping by running

```
> demo("peace")
```

The factual and counterfactual U.N. peacekeeping data sets used in the examples are included in the WhatIf package. You may load them by calling:

```
> data("peacef")
```

and

```
> data("peacecf")
```

which stores them as `R` objects with the corresponding names.

4 Technical Details

The computational task of determining the convex hull membership is made feasible even for large numbers of explanatory variables and observations by the solution proposed in King and Zeng (2006), which eliminates the most time-consuming part of the problem: the characterization of the convex hull itself. In addition, they show that the remaining (implicit) point location problem can be expressed as a linear programming exercise, making it possible to take advantage of existing well-developed algorithms designed for other purposes to speed up the computation. Specifically, a counterfactual x is in the convex hull of the explanatory variables X if there exists a feasible solution to the following standard form linear programming problem:

$$\begin{aligned} &\min\ C'\eta \\ &\text{s.t. } A'\eta = B' \\ &\qquad \eta \geq 0 \end{aligned} \tag{1}$$

where C is a vector of zeros (so that there is no objective function to minimize); η is a vector of coefficients; A' is X' with an additional, final row of 1's; and B' is x' with an additional, final element equal to 1.

The default Gower distance (which is suitable for both quantitative and qualitative data) between a pair of K dimensional points x_i and x_j is defined simply as the average absolute distance between the elements of the two points divided by the range of the data:

$$G_{ij} = \frac{1}{K}\sum_{k=1}^{K}\frac{|x_{ik} - x_{jk}|}{r_k} \tag{2}$$

where the range is $r_k = \max(X_{\cdot k}) - \min(X_{\cdot k})$ and the min and max functions return the smallest and largest elements respectively in the set including the kth element of the explanatory variables X. The optional squared Euclidian distance (which is suitable only for quantitative data) between points x_i and x_j is given by the familiar definition, i.e. the sum of the squared differences between the elements of the two points:

$$E_{ij} = \sum_{k=1}^{K}(x_{ik} - x_{jk})^2\ . \tag{3}$$

5 `R` Function Reference

5.1 Function `whatif()`

This function evaluates your counterfactuals. Specifically, it:

1. Determines if your counterfactuals are in the convex hull of the observed covariate data and are therefore interpolations or if they instead lie outside of it and are therefore extrapolations.

2. Computes the distance from your counterfactuals to each of the n observed data points. The default distance function used is Gower's non-parametric measure.

3. Computes a summary statistic for each counterfactual based on the distances in 2: the fraction of observed covariate data points with distances to your counterfactual less than a value you supply. By default, this value is taken to be the geometric variability of the observed data.

4. Computes the cumulative frequency distribution of each counterfactual for the distances in 2 using values that you supply. By default, Gower distances from 0 to 1 in increments of 0.05 are used.

In other words, this function provides you with both qualitative and quantitative information about your counterfactuals, including two numeric summaries. You can then feed the output of this function either to `plot` to generate a graphical view or to `summary` to get a numerical summary of the results.

5.1.1 Usage

```
whatif(formula = NULL, data, cfact, range = NULL, freq = NULL,
nearby = 1, distance = "gower", miss = "list", choice = "both",
return.inputs = FALSE, return.distance = FALSE, ...)
```

5.1.2 Inputs

formula An optional formula without a dependent variable that is of class "formula" and that follows standard `R` conventions for formulas, e.g. ~ x1 + x2. Allows you to transform or otherwise re-specify combinations of the variables in both `data` and `cfact`. To use this parameter, both `data` and `cfact` must be coercable to data frames; the variables of both `data` and `cfact` must be labeled; and all variables appearing in `formula` must also appear in both `data` and `cfact`. Otherwise, errors are returned. The intercept is automatically dropped. Default is `NULL`.

data May take one of the following forms:

1. A `R` model output object, such as the output from calls to `lm`, `glm`, and `zelig`. Such an output object must be a list. It must additionally have either a `formula` or `terms` component and either a `data` or `model` component; if it does not, an error is returned. Of the latter, `whatif` first looks for `data`, which should contain either the original data set supplied as part of the model call (as in `glm`) or the name of this data set (as in `zelig`), which is assumed to reside in the global environment. If `data` does not exist, `whatif` then looks for `model`, which should contain the model frame (as in `lm`). The intercept is automatically dropped from the extracted *observed covariate data* set if the original model included one.
2. A $n \times k$ non-character (logical or numeric) matrix or data frame of *observed covariate data* with n data points or units and k covariates. All desired variable transformations and interaction terms should be included in this set of k covariates unless `formula` is alternatively used to produce them. However, an intercept should not be. Such a matrix may be obtained by passing model output (e.g., output from a call to `lm`) to `model.matrix` and excluding the intercept from the resulting matrix if one was fit. Note that `whatif` will attempt to coerce data frames to their internal numeric values. Hence, data frames should only contain logical, numeric, and factor columns; character columns will lead to an error being returned.
3. A string. Either the complete path (including file name) of the file containing the data or the path relative to your working directory. This file should be a white space delimited text file. If it contains a header, you must include a column of row names as discussed in the help file for the `R` function `read.table`. The data in the file should be as otherwise described in (2).

Missing data is allowed and will be dealt with via the argument `missing`. It should be flagged using `R`'s standard representation for missing data, `NA`.

cfact A `R` object or a string. If a `R` object, a $m \times k$ non-character matrix or data frame of *counterfactuals* with m counterfactuals and the same k covariates (in the same order) as in `data`. However, if `formula` is used to select a subset of the k covariates, then `cfact` may contain either only these $j \leq k$ covariates or the complete set of k covariates. An intercept should not be included as one of the covariates. It will be automatically dropped from the counterfactuals generated by `Zelig` if the original model contained one. Data frames will again be coerced to their internal numeric values if possible. If a string, either the complete path (including file name) of the file containing the counterfactuals or the path relative to your working directory. This file should be a white space delimited text file. See the discussion under `data` for instructions on dealing with a header. All counterfactuals should be fully observed: if you supply counterfactuals with missing data, they will be list-wise deleted and a warning message will be printed to the screen.

range An optional numeric vector of length k, where k is the number of covariates. Each element represents the range of the corresponding covariate for use in calculating Gower distances. Use this argument when covariate data do not represent the population of interest, such as selection by stratification or experimental manipulation. By default, the range of each covariate is calculated from the data (the difference of its maximum and minimum values in the sample), which is appropriate when a simple random sampling design was used. To supply your own range for the kth covariate, set the kth element of the vector equal to the desired range and all other elements equal to `NA`. Default is `NULL`.

freq An optional numeric vector of any positive length, the elements of which comprise a set of distances. Used in calculating cumulative frequency distributions for the distances of the data points from each counterfactual. For each such distance and counterfactual, the cumulative frequency is the fraction of observed covariate data points with distance to the counterfactual less than or equal to the supplied distance value. The default varies with the distance measure used. When the Gower distance measure is employed, frequencies are calculated for the sequence of Gower distances from 0 to 1 in increments of 0.05. When the Euclidian distance measure is employed, frequencies are calculated for the sequence of Euclidian distances from the minimum to the maximum observed distances in twenty equal increments, all rounded to two decimal places. Default is `NULL`.

nearby An optional scalar indicating which observed data points are considered to be nearby (i.e., within 'nearby' geometric variances of) the counterfactuals. Used to calculate the summary statistic returned by the function: the fraction of the observed data nearby each counterfactual. By default, the geometric variability of the covariate data is used. For example, setting `nearby` to 2 will identify the proportion of data points within two geometric variances of a counterfactual. Default is `1`.

distance An optional string indicating the distance measure to employ. The choices are either `"gower"`, Gower's non-parametric distance measure (G^2), which is suitable for both qualitative and quantitative data; or `"euclidian"`, (squared) Euclidian distance, which is only suitable for quantitative data. The default is the former, `"gower"`.

miss An optional string indicating the strategy for dealing with missing data in the observed covariate data set. `whatif` supports two possible missing data strategies: `"list"`, list-wise deletion of missing cases; and `"case"`, ignoring missing data case-by-case (pairwise deletion). Note that if `"case"` is selected, observations with missing values are still deleted listwise for the convex hull test and for computing Euclidian distances, but pairwise deletion is used in computing the Gower distances to maximally use available information. The user is strongly encouraged to treat missing data using specialized tools such as Amelia prior to feeding the data to `whatif`. Default is `"list"`.

choice An optional string indicating which analyses to undertake. The options are either `"hull"`, only perform the convex hull membership test; `"distance"`, do not perform the convex hull test but do everything else, such as calculating the distance between each counterfactual and data point; or `"both"`, undertake both the convex hull test and the distance calculations (i.e., do everything). Default is `"both"`.

return.inputs A Boolean; should the processed observed covariate and counterfactual data matrices on which all `whatif` computations are performed be returned? Processing refers to internal `whatif` operations such as the subsetting of covariates via `formula`, the deletion of cases with missing values, and the coercion of data frames to numeric matrices. Primarily intended for diagnostic purposes. If `TRUE`, these matrices are returned as a list. Default is `FALSE`.

return.distance A Boolean; should the matrix of distances between each counterfactual and data point be returned? If `TRUE`, this matrix is returned as part of the output; if `FALSE`, it is not. Default is `FALSE` due to the large size that this matrix may attain.

5.1.3 Value

An object of class "whatif", a list containing the following six or seven elements:

call The original call to `whatif`.

inputs A list with two elements, `data` and `cfact`. Only present if `return.inputs` was set equal to `TRUE` in the call to `whatif`. The first element is the processed observed covariate data matrix on which all `whatif` computations were performed. The second element is the processed counterfactual data matrix.

in.hull A logical vector of length m, where m is the number of counterfactuals. Each element of the vector is `TRUE` if the corresponding counterfactual is in the convex hull and `FALSE` otherwise.

dist An $m \times n$ numeric matrix, where m is the number of counterfactuals and n is the number of data points (units). Only present if `return.distance` was set equal to `TRUE` in the call to `whatif`. The $[i, j]$th entry of the matrix contains the distance between the ith counterfactual and the jth data point.

geom.var A scalar. The geometric variability of the observed covariate data.

sum.stat A numeric vector of length m, where m is the number of counterfactuals. The mth element contains the summary statistic for the corresponding counterfactual. This summary statistic is the fraction of data points with distances to the counterfactual less than `nearby*gv`, which by default is the geometric variability of the covariates.

cum.freq A numeric matrix. By default, the matrix has dimension $m \times 21$, where m is the number of counterfactuals; however, if you supplied your own frequencies via the argument `freq`, the matrix has dimension $m \times f$, where f is the length of `freq`. Each row of the matrix contains the cumulative frequency distribution for the corresponding counterfactual calculated using either the distance measure-specific default set of distance values or the set that you supplied (see the discussion under the argument `freq`). Hence, the $[i, j]$th entry of the matrix is the fraction of data points with distances to the ith counterfactual less than or equal to the value represented by the jth column. The column names contain these values.

5.2 Function `plot.whatif()`

This function generates a cumulative frequency plot of distances, graphically summarizing the distance of your counterfactuals from the data. It takes as its input an object returned by the function `whatif`, which has class "whatif". It is called by the generic function `plot`. The cumulative frequencies appear on the vertical axis. They are the fraction of rows in the observed data set with either Gower or (squared) Euclidian distances to the counterfactuals less than or equal to the given value on the horizontal axis. Counterfactuals in the convex hull are plotted with a solid line and counterfactuals outside of the convex hull with a dashed line.

5.2.1 Usage

```
plot(x, type = "f", numcf = NULL, eps = FALSE, ...)
```

5.2.2 Inputs

x An object of class "whatif", the output of the function `whatif()`.

type A character string; the type of plot of the cumulative frequencies of the distances to be produced, including: `"f"` for cumulative frequencies only; `"l"` for LOWESS smoothing of cumulative frequencies only; `"b"` for both cumulative frequencies and LOWESS smoothing. LOWESS scatterplot smoothing is plotted in blue and the unsmoothed frequencies in black. Default is `"f"`.

numcf A numeric vector; the specific counterfactuals to be plotted. Each element represents a counterfactual, specifically its row number from the matrix or data frame of counterfactuals. By default, all counterfactuals are plotted. Default is `NULL`.

eps A Boolean; should an encapsulated postscript file be generated? Setting the argument equal to `TRUE` generates an `.eps` file, which is saved to your working directory with file name of form '`graph_'type'_'numcf'.eps`', where `'type'` and `'numcf'` are the values of the respective arguments. Specifically, `'numcf'` takes the value of the first element of the argument `numcf` unless all counterfactuals were plotted, in which case `all` appears in the place of `'numcf'`. Default is `FALSE`, which instead prints the graph to the screen.

5.2.3 Value

Either a graph printed to the screen or an encapsulated postscript file saved to your working directory. In the latter case, the file name has form '`graph_'type'_'numcf'.eps`', where `'type'` and `'numcf'` are the values of the respective arguments.

5.3 Function `print.whatif()`

This function prints the information generated by a call to `whatif` to the screen. It takes as its input an object returned by the function `whatif`, which has class "whatif". It is called by the generic function `print`.

5.3.1 Usage

```
print(x, print.dist = FALSE, print.freq = FALSE, ...)
```

5.3.2 Inputs

x An object of class "whatif", the output of the function `whatif`.

print.dist A Boolean; should the matrix of pairwise distances between each counterfactual and data point be printed to the screen, if it was returned? Default is `FALSE`.

print.freq A Boolean; should the matrix of cumulative frequencies of distances for each counterfactual be printed to the screen? Default is `FALSE`.

5.3.3 Value

A printout to the screen of the information contained in the `whatif` output object.

5.4 Function `print.summary.whatif()`

This function prints the information generated from the `whatif` output object by a call to `summary` to the screen. It takes as its input an object returned by the function `summary.whatif`, which has class "summary.whatif". It is called by the generic function `print`.

5.4.1 Usage

```
print(x, ...)
```

5.4.2 Inputs

x An object of class "summary.whatif", the output of the function `summary.whatif`.

5.4.3 Value

A printout to the screen of the `whatif` information summarized in the `summary.whatif` output object.

5.5 Function `summary.whatif()`

This function summarizes the information produced by `whatif` about your counterfactuals. It takes as its input the object returned by the function `whatif`, which has class "whatif". It is called by the generic function `summary`. The summary generated is returned as well as printed to the screen.

5.5.1 Usage

```
summary(object, ...)
```

5.5.2 Inputs

object An object of class "whatif", the output of the function `whatif`.

5.5.3 Value

An object of class "summary.whatif", a list containing the following five elements:

call The original call to `whatif`.

m A scalar. The total number of counterfactuals evaluated.

m.inhull A scalar. The number of counterfactuals evaluated that are in the convex hull of the observed covariate data.

mean.near A scalar. The average percentage of data 'nearby' each counterfactual, where the average is taken over all counterfactuals.

sum.df A data frame with three columns and m rows, where m is the number of counterfactuals. The first column, `cfact`, indexes the counterfactuals. The second column, `in.hull`, contains the results of the convex hull test. The third column, `per.near`, contains the percentage of data points nearby each counterfactual.

This object is printed to the screen.

References

Ho, Daniel, Kosuke Imai, Gary King and Elizabeth Stuart. 2007. "Matching as Nonparametric Preprocessing for Reducing Model Dependence in Parametric Causal Inference." *Political Analysis* 15:199–236. http://gking.harvard.edu/files/abs/matchp-abs.shtml.

Imai, Kosuke, Gary King and Olivia Lau. 2006. "Zelig: Everyone's Statistical Software.". http://gking.harvard.edu/zelig.

Imai, Kosuke, Gary King and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development." *Journal of Computational Graphics and Statistics* 17(4):1–22. http://gking.harvard.edu/files/abs/z-abs.shtml.

King, Gary and Langche Zeng. 2006. "The Dangers of Extreme Counterfactuals." *Political Analysis* 14(2):131–159. http://gking.harvard.edu/files/abs/counterft-abs.shtml.

King, Gary and Langche Zeng. 2007. "When Can History Be Our Guide? The Pitfalls of Counterfactual Inference." *International Studies Quarterly* (March):183–210. http://gking.harvard.edu/files/abs/counterf-abs.shtml.