Gradient boosted weighting for linkage failures in US Census Bureau datasets

Matthew Cefalu, John Sullivan, Narayan Sastry, and Elizabeth Fussell

2021-04-05

Introduction

The **twangRDC** R package is a streamlined version of the `twang` R package that was created specifically for use in the Federal Statistical Research Data Centers (FSRDCs), which are restricted data enclaves run by the Census Bureau to promote research use of data products from the Census and other federal statistical agencies. In particular, the **twangRDC** package contains functions to address linkage failures, which may systematically affect some records creating bias in population estimates. Further, the same functions can be used to ensure that a comparison group is equivalent to a treatment group on observable characteristics.

The package utilizes gradient boosted models to estimate weights for linkage failures and comparison group construction. Results using **twangRDC** will not necessarily be reproduced using `twang` due to important differences in implementation. The **twangRDC** package allows for much larger datasets, like those used in demographic linkage, and many more covariates, but users should note that with smaller datasets, the `twang` package is computationally more efficient. The **twangRDC** package uses *xgboost* for gradient boosting, which provides user with an optimized gradient boosting library that can utilize parallel computation within the FSRDCs.

Methodology

The **twangRDC** package uses gradient boosted models to derive weights for nonequivalent groups. The algorithm alternates between adding iterations to the gradient boosted model (increasing its complexity) and evaluating balance between the two groups on the covariates. The algorithm automatically stops when additional iterations no longer improve balance. The package allows the user to generate weights for two different scenarios: linkage failures and the construction of a comparison group.

Linkage failures

Protected Identification Keys (PIKs) are linking keys used by the Census Bureau to match person records across datasets[1]. For example, using a PIK, a respondent in the 2000 Decennial Census can be linked to their response in the 2010 Decennial Census. PIKs are assigned through a probabilistic matching process that links survey or census records containing Personally Identifiable Information (PII) like name, address, date of birth and place of birth, to a Social Security reference file.

Not all survey and census records can be linked to the reference file and as such not all records will be assigned a PIK. Linkage failures may occur if the PII used to match to the reference file is of insufficient quality (e.g., missing information), has insufficiently unique identifying information (e.g., a common name such as John Smith), or if an individual has no matching record in the Social Security reference file (e.g., undocumented residents). PIKs can be assigned for the large majority of 2000 Decennial Census records. A

[1] Wagner, Deborah and Mary Layne. The Person Identification Validation System (PVS): Applying the Center Administrative Records Research and Applications' (CARRA) Record Linkage Software. US Census Bureau, Center Administrative Records Research and Applications Working Paper #2014-01. 2014.

Decennial Census record must have a PIK in order to link it to administrative records that contain other information, such as mortality or annual residence.

The **twangRDC** pacakge can generate weights to account for linkage failures such that observations with PIKs are representative of the population. It does this by using standard nonresponse weights, where nonresponse is defined by linkage failure. This is achieved by weighting observations with PIKs by the inverse probability of having a PIK. As a note, although designed for linkage failure, the **twangRDC** can be used for other nonresponse and missing data.

The steps of the algorithm are described here assuming the user specifies strata in the model, but the steps are the same without strata by considering the data coming from a single stratum. Guidance on specific decisions relevant for each step is provided throughout the examples.

1. Calculate the population-level means of the covariates within strata. If weights are specified by the user, the population-level means are calculated accounting for the weights.
2. Initialize a gradient boosted model that includes all covariates and the strata as a factor.
3. Add `iters.per.step` iterations to the gradient boosted model.
4. Calculate the means of the covariates within strata among observations with PIK using weights based on the current model.
5. Calculate the standardized differences of the covariates between weighted observations with PIK versus the population within strata as calculated in step #1.
6. Summarize the model fit by taking the maximum of the absolute standardized differences within strata.
7. Check if the average of the maximum absolute standardized differences across strata has been minimized. If a minimum has been achieved, stop. Otherwise, return to step #3.

Comparison group construction

If the goal is the construction of a comparison group, **twangRDC** can apply gradient boosted propensity score weights for the average treatment effect on the treated. Treatment observations receive a weight of 1, while comparison observations receive a weight of the odds of treatment based on the propensity score model. The steps of the algorithm are described here assuming the user specifies strata in the model, but the steps are the same without strata by considering the data coming from a single stratum. Guidance on specific decisions relevant for each step is provided throughout the examples.

1. Calculate the treatment group means of the covariates within strata. If weights are specified by the user, the treatment group means are calculated accounting for the weights.
2. Initialize a gradient boosted model that includes all covariates and the strata as a factor.
3. Add `iters.per.step` iterations to the gradient boosted model.
4. Calculate the means of the covariates within strata among comparison cases using propensity score weights based on the current model.
5. Calculate the standardized differences of the covariates between the treatment and propensity score weighted comparison group means.
6. Summarize the model fit by taking the maximum of the absolute standardized differences within strata.
7. Check if the average of maximum absolute standardized differences across strata has been minimized. If a minimum has been achieved, stop. Otherwise, return to step #3.

Description of arguments

The primary function of **twangRDC** is `ps.xgb`, which performs the methodology previously described. In Table 1, we describe the options of `ps.xgb`. Additional details can be found in the help file.

Table 1: Table 1: Description of arguments to the ps.xgb function

Argument	Description
formula	A symbolic description of the model to be fit with an observed data indicator or a treatment indicator on the left side of the formula and the variables to be balanced on the right side. If `strata` is specified, the model automatically includes the strata as a factor variable on the right hand side of the equation.
linkage	An indicator of whether the weighting should be for linkage failure or comparison group construction. `linkage=TRUE` requests weighting to account for linkage failure, while `linkage=FALSE` requests weighting for comparison group construction.
strata	An optional factor variable identifying the strata. If specified, balance is optimized within strata.
data	The data.
params	*xgboost* parameters. Details below.
file	An optional filename to save intermediate results. The file is overwritten at each step of the algorithm.
iters.per.step	An integer specifying the number of iterations to add to the model at each step of algorithm.
max_steps	An integer specifying the maximum number of steps to take while optimizing the gradent boosted model. The maximum number of iterations considered during optimization is given by `max_steps*iters.per.step`.
id.var	A variable that uniquely identifies observations.
min.width	An integer specifying the minimum number of iterations between the current number of model iterations and the optimal value.
save.model	A logical value indicating if the *xgboost* model be saved as part of the output object.
weights	An optional variable that identifies user defined weights to be incorporated into the optimization, e.g., sampling design weights. If specified, the output automatically accounts for these weights.

Description of *xgboost* parameters

A detailed description of the *xgboost* options can be found in the *xgboost* documentation. Here, we briefly describe the options that are most useful in this context.

- `eta` is a value between 0 and 1 that controls the learning rate. Smaller values of `eta` reduce overfit but require additional iterations to achieve optimal balance.
- `max_depth` is the maximum allowable depth of the gradient boosted trees. A larger values allows the model to consider more complex interactions between the covariates. Increasing `max_depth` may require a reduction in `eta`.
- `min_child_weight` is the minimum number of observations needed to further partition a tree. If a leaf node is such that it has fewer than `min_child_weight` observations, the tree partitioning process will stop. Users can conceptualize this as similar to the number of observations in a level of a categorical variable for a standard regression model. Note, if weights are specified in `ps.xgb`, then `min_child_weight` references the sum of the weights.

Selection of argument values

The options for **twangRDC** should be set to achieve optimal balance. Unfortunately, many of the arguments are tuning parameters that should be tweaked to explore if the resulting model achieves better or worse balance than other choices of the arguments. We recommend that users run the models several times modifying the choice of the arguments to optimize their choice. In most cases, we have found that small changes to the arguments will not result in substantial differences in the underlying model fit. However, it is important to explore this possibility for each new analysis. Some general recommendations are provided here.

- The covariates listed in the `formula` should include all covariates that are to be balanced. If working with small to moderate sample sizes, it is likely that achieving balance on a large set of covariates is not feasible, and the user should make the choice of covariates as parsimonious as possible. The underlying gradient boosted model will automatically consider interactions and nonlinearities, but balance is only calculated based on the form of the covariates in the `formula`. To assess balance for specific interactions, the interactions must be included in the `formula`.
- A larger value of `iters.per.step` will reduce the computation time necessary for the model to finish. However, there is a tradeoff between the choice of `iters.per.step` and the balance achieved by the model. Larger values will result in worse balance. We recommend values between 50 and 500 for larger datasets.
- `eta`, `max_depth`, and `min_child_weight` should be set to a combination of values that allow the *xgboost* model to run for at least a few thousand iterations. If the optimum iteration is too small, users can decrease `eta` or `max_depth` or increase `min_child_weight`. Conversely, if the algorithm does not stop within the users expectations, the value of `eta` or `max_depth` can be increased or `min_child_weight` decreased. Our experience suggests that values of `eta` between 0.1 and 0.3 are a good place to start.

Example use of twangRDC

We will highlight two uses of the **twangRDC** R package. First, we will generate weights that ensure individuals with PIKs are representative of their geographic region's population. Second, we will highlight using the **twangRDC** R package to generate propensity score weights such that a group of southern metropolitan areas can be used as a comparison group for residents of Orleans Parish, Louisiana, which corresponds exactly with the boundaries of the City of New Orleans.

The Data

A simulated data file was created for use in this vignette. It contains simulated records for residents of Orleans Parish and other metropolitan areas in the South census region. We built the file exclusively from public use data, but it mirrors the structure of restricted versions of the 2000 Decennial Census available through the FSRDC network[2]. Each simulated record includes basic individual demographic characteristics and basic household characteristics. Each record also includes two important indicators, one to simulate whether the individual record received a PIK and another to denote whether the individual lived in Orleans Parish.

The data was created by extracting all "short form" variables for households and individuals from the 5% Integrated Public Use Microdata Sample (IPUMS[3]) of the 2000 Decennial Census for Orleans Parish and for a selection of other southern metropolitan areas. We simulated assignment of households to census tracts and attached tract identifiers and characteristics.

We also simulated an indicator of PIK assignment (PIK=yes/no) to person records. Public use data do not include PIK assignment, so we estimated a predicted probability of receiving a PIK by using estimated regression parameters from Bond et al. 2013[4]. We added a random error to the predicted probability so that

[2]https://www.census.gov/fsrdc

[3]Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset]. Minneapolis, MN: IPUMS, 2020. https://doi.org/10.18128/D010.V10.0

[4]Bond, Brittany, J. David Brown, Adela Luque and Amy O'Hara. The nature of the bias when studying only linkable person records: Evidence from the ACS. US Census Bureau, Center Administrative Records Research and Applications Working Paper #2014-08, 2014.

the PIK assignment status **is not deterministic** and converted the predicted probability into a dichotomous PIK=yes/no variable.

Lastly, we pooled Orleans Parish records with those from other southern metropolitan areas, create an indicator for Orleans Parish residence and, for the purposes of this vignette, sampled the data to shrink the size of the dataset. The simulated file contains individual records for 8,893 residents of Orleans Parish, LA and 9,503 individual records for residents from select southern metropolitan areas.

Table 2 provides a description of the data element included in the simulated data file.

Table 2: Table 2: Description of data elements

Data element	Description	Labels and Coding
metarea	A categorical variable for metropolitan area.	Atlanta, GA (52) Memphis, TN/AR/MS (492) New Orleans, LA (556) Washington, DC/MD/VA (884)
c00_age12	A categorical variable for age in years at the 2000 Decennial Census.	0 to 2 years old (1)3 to 5 years old (2)6 to 9 years old (3)10 to 14 years old (4) 15 to 18 years old (5) 19 to 24 years old (6) 25 to 34 years old (7) 35 to 44 years old (8) 45 to 54 years old (9) 55 to 64 years old (10) 65 to 74 years old (11) 75 and older (12)
c00_sex	A binary indicator of sex as reported on the 2000 Decennial Census.	Male (0) Female (1)
c00_race	A categorical variable for race as reported on the 2000 Decennial Census.	White (1)African American (2)American Indian or Alaskan Native (3)Asian (4)Other Asian or Pacific Islander (5)Some other race (6)
c00_nphu	The number of persons in housing unit as reported on the 2000 Decennial Census.	1 to 16
hhid	Household identifier.	
tract_id_str	Census tract identifier.	
sim_pik	Simulated binary indicator of PIK assignment.	No PIK assigned (0)PIK assigned (1)
nola_rec	Binary indicator for record from Orleans Parish.	Not Orleans Parish Record (0)Orleans Parish Record (1)
id	An ID variable that uniquely identifies individuals in the dataset.	

Weighting to account for linkage failure

As previously mentioned, we will generate weights that ensure individuals with PIKs are representative of their geographic region. To keep the computational time of this vignette down, we focus only on a subset of Orleans parish. First, we load the **twangRDC** package and our simulated dataset.

```
library(twangRDC)
data(nola_south)
```

Next, it is important that the variables of the dataset are coded as intended. In this case, we convert several of variables to factors.

```
# factors need to be coded as such
nola_south$metarea = as.factor(nola_south$metarea)
nola_south$c00_age12 = as.factor(nola_south$c00_age12)
nola_south$c00_race = as.factor(nola_south$c00_race)
nola_south$c00_sex = as.factor(nola_south$c00_sex)
```

In a final data preparation step, we limit the dataset to Orleans Parish.

```
# only consider Orleans parish
nola_only = subset(nola_south , metarea==556)
```

In this case, we wish to generate weights to ensure that individuals with PIKs are representative of their entire Census tract. That is, for each Census tract, we want to generate weights such that the observations with PIKs within the Census tract are representative of the tract's population. This is achieved by specifying `linkage=TRUE`, which tells the function that we wish to generate nonresponse weights based on linkage failure, and `strata="tract_id_str"`, which tells the function that we wish to generate weights that are representative within strata defined by Census tracts. The formula `sim_pik ~ c00_age12 + c00_race + c00_sex` identifies the observations with PIK on the left hand side and the characteristics that we want to consider when estimating the weights the right hand side.

```
# set the model parameters
params = list(eta = .1 , max_depth = 5 , min_child_weight=50)

# fit the xgboost model
res.pik = ps.xgb(sim_pik ~ c00_age12 + c00_race + c00_sex ,
             strata="tract_id_str",
             data=nola_only,
             params=params,
             max.steps=50,
             iters.per.step=100,
             min.iter=1000,
             id.var="id",
             linkage = TRUE)
```

The parameters of the underlying *xgboost* model are specified in `params`. These were described in more detail in a previous section. The `id.var="id"` provides a unique identifier for observations such that the generated weights can easily be merged back in with the original data. The other options specified in the `ps.xgb` function control how frequently the algorithm checks for convergence and how many iterations should be considered before stopping. First, `iters.per.step=100` tells the algorithm to only consider every 100-th iteration when evaluating convergence. Larger values improve computational time by reducing the number of balance evaluations, while smaller values may achieve slightly better balance. Next, `min.iter=1000` tells the algorithm that at least 1000 iterations must be used before stopping for convergence. Larger values ensure that more complex models are evaluated before determining the optimal set of weights. Finally, `max.steps=50` indicates that the algorithm should only evaluate balance up to 50 times before stopping. The maximum number of iterations of the *xgboost* model is given by `max.steps*iters.per.setp`, which in this case is 5000. In general, this value should be large to ensure that the optimum set of weights is achieved. The default value is `max.steps=Inf`, which will continue adding iterations to the model until the convergence criteria is met. Due to computational concerns, we recommend testing your code with values of `iters.per.step` and `max.steps` such that the total number of iterations is small (1000 to 10000). Once you have determined the model is working as intended, set `max.steps` to `Inf`.

Now that the weights have been estimated, we will evaluate their quality. First, we need to ensure that a sufficient number of iterations have been used such that the balance criteria is minimized.

```
plot(res.pik)
```

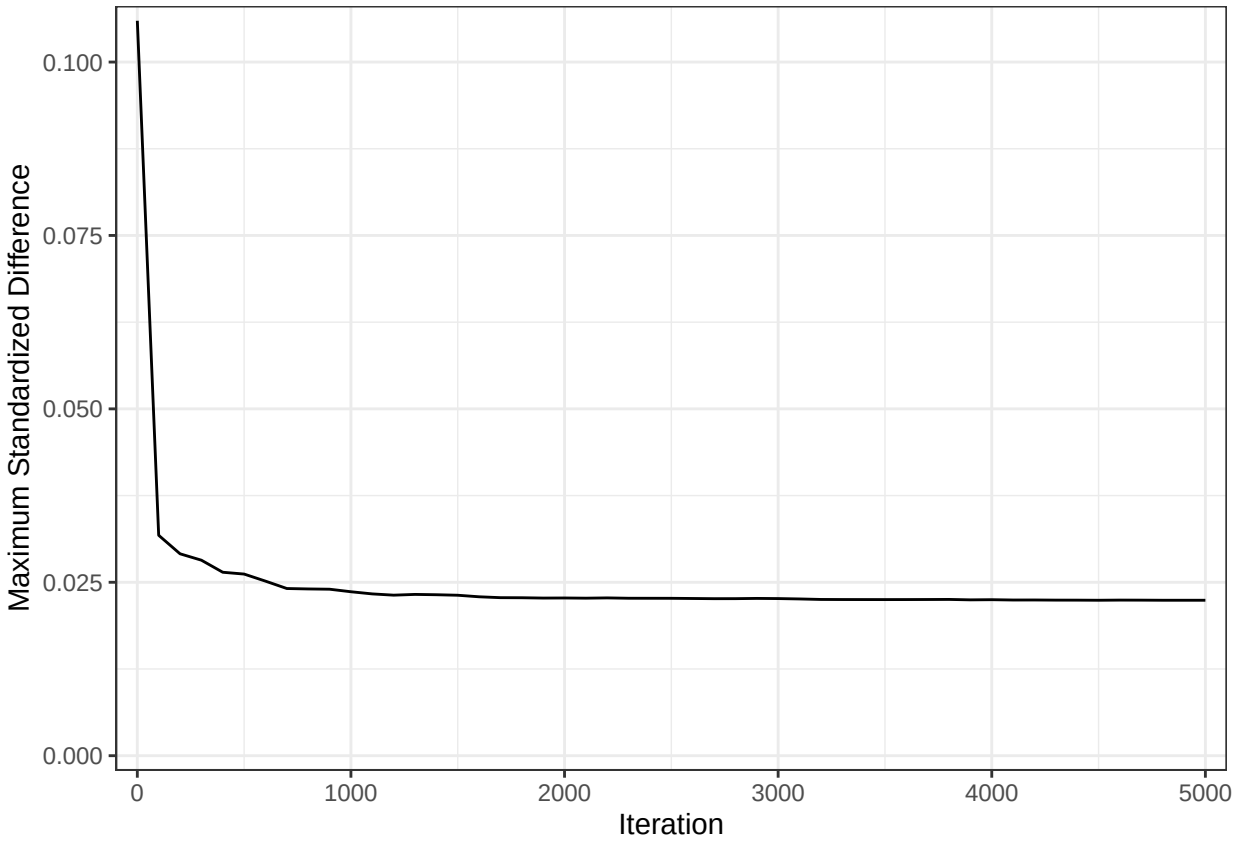

Figure 1: Figure 1: Convergence of gradient boosted model for linkage failure.

The `plot` function provides a plot of the balance criteria, which in this case is the average of the strata-specific maximum absolute standardized difference of the covariates, versus the number iterations. As shown in this figure, we are verifying that the algorithm has run for a sufficient number of iterations such that a clear minimum has been achieved.

After evaluating convergence of the algorithm, we can access the quality of the weights using balance tables. First, the `bal.table` function will produce the population mean for each covariate, as well as the unweighted and the weighted mean among those with PIK. It also provides the standardized difference for each covariate before and after weighting. The goal is for the standardized differences after weighting to all be small, with many researchers recommending absolute values less than 0.1 as the target. Regardless of the specific target, standardized differences close to zero are ideal, and the quality of the balance should be assessed with the context of the specific analysis in mind. In this case, all of our standardized differences are very close to zero.

```
bal.table(res.pik)
```

	Population Mean	Unadjusted Mean	Adjusted Mean	Unadjusted Standardized Difference	Adjusted Standardized Difference
c00_age12:1	0.032	0.017	0.030	0.086	0.011
c00_age12:2	0.037	0.023	0.037	0.073	0.000
c00_age12:3	0.036	0.025	0.035	0.059	0.004
c00_age12:4	0.066	0.056	0.066	0.042	0.000
c00_age12:5	0.060	0.065	0.060	-0.019	-0.001
c00_age12:6	0.059	0.054	0.059	0.022	-0.001

	Population Mean	Unadjusted Mean	Adjusted Mean	Unadjusted Standardized Difference	Adjusted Standardized Difference
c00_age12:7	0.126	0.127	0.126	-0.004	-0.001
c00_age12:8	0.174	0.179	0.174	-0.015	-0.001
c00_age12:9	0.176	0.192	0.176	-0.042	-0.001
c00_age12:10	0.112	0.126	0.112	-0.044	-0.001
c00_age12:11	0.063	0.070	0.063	-0.028	-0.001
c00_age12:12	0.059	0.066	0.059	-0.032	-0.002
c00_race:1	0.604	0.647	0.605	-0.088	-0.003
c00_race:2	0.364	0.320	0.363	0.091	0.003
c00_race:3	0.000	0.000	0.000	-0.001	0.000
c00_race:4	0.005	0.005	0.005	-0.009	0.000
c00_race:5	0.012	0.013	0.012	-0.004	0.001
c00_race:6	0.015	0.015	0.015	0.001	0.001
c00_sex:0	0.512	0.511	0.511	0.003	0.003
c00_sex:1	0.488	0.489	0.489	-0.003	-0.003

Next, balance can be assessed within Census tract since our goal for this model was to generate weights such that observations with PIKs are representative of their Census tract. The `type='strata'` option tells `bal.table` to provide the maximum absolute standardized difference by the strata variable, in this case, the maximum absolute standardized difference of the covariates within each Census tract. We additionally specify `include.var=TRUE`, which identifies the covariate that the maximum absolute standardized difference corresponds to within each Census tract, `decreasing = T`, which orders the strata in decreasing order of their weighted standardized differences, and `n=3`, which only prints the three strata with the largest absolute standardized differences. Note that this table produces a single row for each stratum, such that the table only represents the covariate with the worst balance for each stratum.

```
bal.table(res.pik , type='strata' , include.var=TRUE , n=3 , decreasing = T)
```

	Stratum	Unadjusted Maximum Standardized Difference	Adjusted Maximum Standardized Difference	Variable
3	22071012101	0.114	0.083	c00_age12:1
1	22071001741	0.136	0.006	c00_race:5
2	22071011400	0.091	0.000	c00_race:6

This table allows us to assess which strata had the worst balance after weighting, and among those strata, which covariates were problematic. In this case, both our within strata balance and our overall population balance look excellent. We conclude that the weighting has achieved our goal of making the sample representative of the population.

Extracting weights

The `get.weights` function extracts the weights at the optimal iteration. The resulting data contains the weights and the ID variable specified in `id.var`. The weights can then be merged back in with the original data using the `id.var`. Note that base R merge function is slow compared to modern alternatives. If your data is large, consider `data.table` or `dplyr`.

```
# extract weights
w = get.weights(res.pik)

# merge weights into data
dta=merge(dta , w , by='id' , all=TRUE)
```

These weights can be used in subsequent steps of the analysis. For example, if matching a specific marginal population total is necessary, the weights output by this function can be used as an input to a post-stratification step. Alternately, if the goal of the study is to compare two groups, these weights can be used as inputs into a propensity score model.

Comparison group construction

Our second example use of **twangRDC** will generate a comparison group for Orleans Parish consisting of residents of other southern metropolitan areas. The steps of the process remain the same as the PIK weighting example, but with minor adjustments to the interpretation and specification of the model. First, `linkage = FALSE` specifies that we are no longer interested in weights to account for linkage failure, but instead, we wish to generate a comparison group using propensity score weights. Note that `ps.xgb` only estimates the average treatment effect on the treated, with the treatment group identified by records with a value of 1 in the left hand side of the formula. Our formula now includes an interaction between age and sex. The informs the algorithm to evaluate balance based on the joint distribution of age and sex. Currently, **twangRDC** only allows for specification of interactions between factor variables. Finally, we no longer specify a stratification variable, as we are attempting to create a comparison group that is representative of Orleans parish as a whole. If we were interested in using the linkage failure weights from our previous example, they could be specified in the `weights` option, and the output of this model would automatically account for those weights.

```
# set the model parameters
params = list(eta = .3 , max_depth = 5 , subsample = 1 ,
              max_delta_step=0 , gamma=0 , lambda=0 , alpha=0,
              min_child_weight=50 , objective = "binary:logistic")

# fit the xgboost model
res.ps = ps.xgb(nola_rec ~ c00_age12:c00_sex + c00_race ,
             data=nola_south,
             params=params,
             max.steps=25,
             iters.per.step=100,
             min.iter=1000,
             min.width = 500,
             id.var="id",
             linkage = FALSE)
```

Following the model fit, the same steps covered in the linkage failure example should be used here as well. First, evaluate the convergence of the model using the `plot` function. We are verifying that the algorithm has run for a sufficient number of iterations such that a clear minimum has been achieved.

```
plot(res.ps)
```

Next, we evaluate the performance of the algorithm by looking at balance tables. Since we specified `linkage=FALSE`, the `bal.table` function will now produce the treatment group mean for each covariate, as well as the mean before and after propensity score weighting among comparison cases. It also provides the standardized differences for each covariate. As in the linkage failure example, the goal is for the standardized differences after weighting to all be close to zero.

```
bal.table(res.ps)
```

	Treatment Group Mean	Unadjusted Comparison Group Mean	Adjusted Comparison Group Mean	Unadjusted Standardized Difference	Adjusted Standardized Difference
c00_race:1	0.6037	0.5943	0.6044	0.0192	-0.0014
c00_race:2	0.3639	0.2960	0.3647	0.1411	-0.0018
c00_race:3	0.0001	0.0018	0.0001	-0.1581	-0.0010

	Treatment Group Mean	Unadjusted Comparison Group Mean	Adjusted Comparison Group Mean	Unadjusted Standardized Difference	Adjusted Standardized Difference
c00_race:4	0.0046	0.0084	0.0043	-0.0562	0.0039
c00_race:5	0.0124	0.0394	0.0118	-0.2442	0.0055
c00_race:6	0.0153	0.0601	0.0146	-0.3650	0.0056
c00_age12..c00_sex:1	0.0189	0.0238	0.0192	-0.0359	-0.0026
c00_age12..c00_sex:1	0.0135	0.0206	0.0137	-0.0618	-0.0018
c00_age12..c00_sex:2	0.0233	0.0235	0.0234	-0.0013	-0.0009
c00_age12..c00_sex:2	0.0141	0.0224	0.0134	-0.0710	0.0056
c00_age12..c00_sex:3	0.0207	0.0330	0.0207	-0.0868	-0.0004
c00_age12..c00_sex:3	0.0151	0.0303	0.0151	-0.1251	-0.0005
c00_age12..c00_sex:4	0.0364	0.0390	0.0365	-0.0139	-0.0003
c00_age12..c00_sex:4	0.0297	0.0356	0.0297	-0.0347	-0.0002
c00_age12..c00_sex:5	0.0262	0.0276	0.0262	-0.0086	0.0003
c00_age12..c00_sex:5	0.0340	0.0250	0.0336	0.0492	0.0018
c00_age12..c00_sex:6	0.0332	0.0360	0.0333	-0.0157	-0.0007
c00_age12..c00_sex:6	0.0256	0.0385	0.0257	-0.0815	-0.0006
c00_age12..c00_sex:7	0.0685	0.0811	0.0686	-0.0501	-0.0003
c00_age12..c00_sex:7	0.0576	0.0873	0.0576	-0.1278	0.0000
c00_age12..c00_sex:8	0.0825	0.0843	0.0826	-0.0064	-0.0001
c00_age12..c00_sex:8	0.0912	0.0890	0.0913	0.0075	-0.0005
c00_age12..c00_sex:9	0.0887	0.0689	0.0887	0.0696	-0.0001
c00_age12..c00_sex:9	0.0871	0.0710	0.0869	0.0571	0.0007
c00_age12..c00_sex:1	0.0566	0.0397	0.0565	0.0731	0.0001
c00_age12..c00_sex:1	0.0554	0.0418	0.0555	0.0597	-0.0001
c00_age12..c00_sex:1	0.0292	0.0207	0.0293	0.0505	-0.0006
c00_age12..c00_sex:1	0.0338	0.0268	0.0334	0.0388	0.0022
c00_age12..c00_sex:1	0.0278	0.0132	0.0278	0.0890	-0.0004
c00_age12..c00_sex:1	0.0309	0.0207	0.0310	0.0589	-0.0003

Since no strata were specified in the model, the `type='strata'` option used in the linkage failure example is no longer useful. These weights can be extracted and merged in with the data using the same steps described in the linkage failure example.

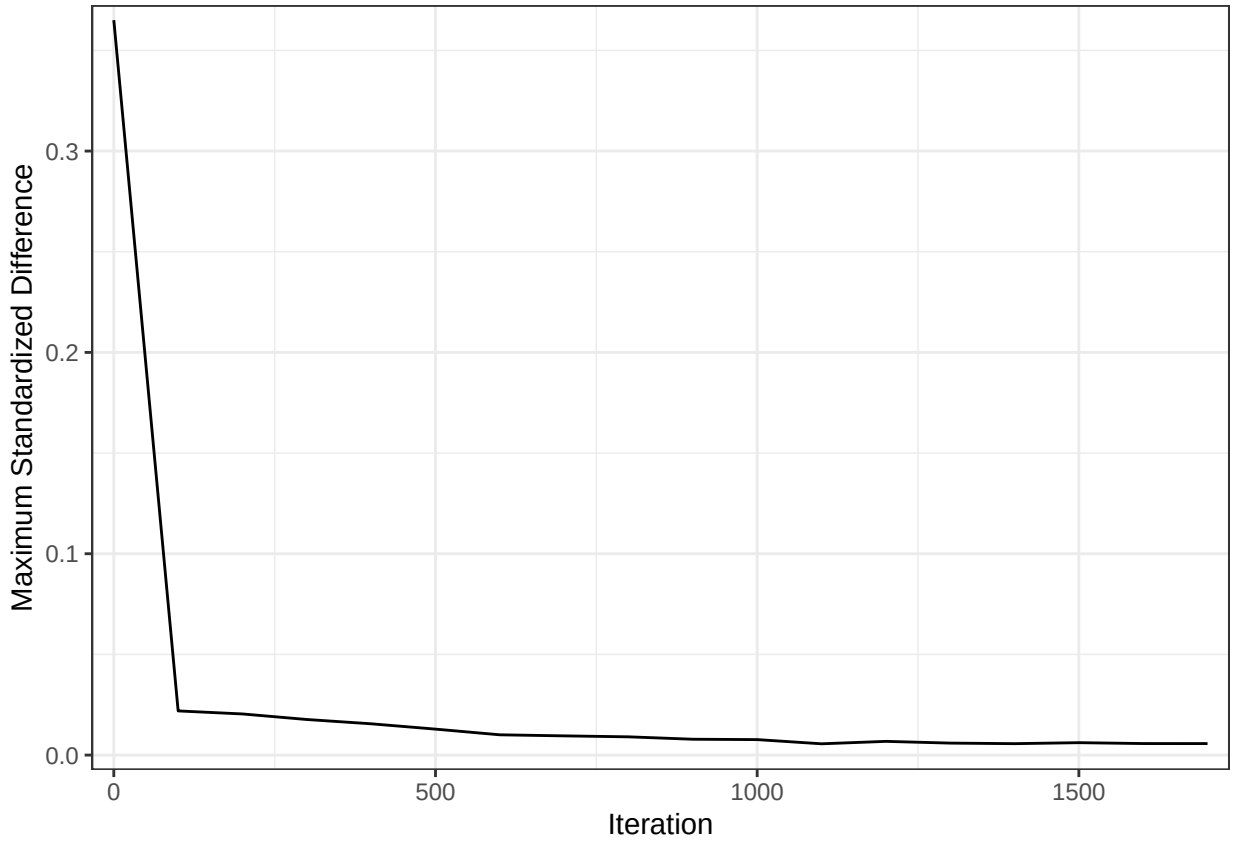

Figure 2: Figure 2: Convergence of gradient boosted model for comparison group construction.