IMPS Process Data Workshop: Session 4 (Applications) Handout

Susu Zhang

7/19/2021

Application 1: Predictions

In this section, we use the MDS features extracted from process data to make predictions about dichotomous or continuous variables. Examples of variables that can be predicted include:

- Final response: correct/incorrect, polytomous score
- Overall proficiency: uni- or multidimensional θ for response accuracy
- External covariates: demographics, other cognitive/noncognitive traits

We denote a (1-dimensional) predicted variable by Y. Y can be continuous or categorical.

The MDS process features (denoted X) are the $K-$dimensional numerical features, which are returned by running `seq2feature_mds` on the problem solving processes. The illustrations are based on MDS features, but the methods are equally applicable to other types of features, such as those returned from `seq2feature_seq2seq`.

Method: Ridge Regression

The method we use for these predictions is ridge regression. The dimension of the process features (K) can be high, and it is possible that only a small number of dimensions are related to the predicted variable. Unlike Ordinary Least Squares (OLS) regression, which minimizes the residual sum of squares (RSS), ridge regression adds an additional L_2 penalty to the regression coefficients, which helps prevent overfitting.

For the prediction of a continuous variable, the regression coefficients (βs) minimize

$$\sum_{i=1}^{N}\left(Y_i-\beta_0-\sum_{k=1}^{K}\beta_k X_{ik}\right)^2+\lambda\sum_{k=1}^{K}\beta_k^2=RSS+\lambda\sum_{k=1}^{K}\beta_k^2.$$

For the prediction of a dichotomous variable, the coefficients (βs) minimize

$$-\frac{1}{N}\log L(Y;\mathbf{X},\boldsymbol{\beta})+\lambda\sum_{k=1}^{K}\beta_k^2.$$

Here, λ is the penalty term. Larger λs give more penalty to the regression coefficients, which will lead to smaller coefficients. In practice, λ can be selected using cross validation (CV).

Ridge regression (and the selection of λ using CV) can be implemented using the `glmnet` R package. The package can be installed by running `install.packages('glmnet')`. Below, we illustrate this with two examples based on the PISA 2012 data.

Example: Predictions with Climate Control Processes

Here, we present 2 examples using MDS process features for predictions, one with a continuous variable, and the other with a dichotomous variable.

The two predicted variables (Ys) are as follows:

- Overall CPS proficiency (mean plausible value): Continuous.
- Score (0/1) on climate control item: Dichotomous.

For both examples, our predictor variables (X) are the $K = 50-$dimensional MDS features extracted from the climate control processes. The MDS features have been pre-trained using the `seq2feature_mds` function. For the first prediction (overall CPS proficiency), we additionally include a 51^{st} dimension, which is the final response accuracy (`cc_data$responses`).

The PISA CPS proficiency variable can be found from the public PISA 2012 data (http://www.oecd.org/pisa/pisaproducts/database-cbapisa2012.htm).

The MDS features (`mds_fts`), binary final responses (`resp_cc`), and the overall CPS proficiency (`PVCPRO`) are available in the `pred_example.RData` file.

To load the PISA example data to your R environment, run `load('pred_example.RData')`. We can first look at some basic information and descriptive statistics of the features and the Ys.

```
# MDS feature
cat('Object class of mds_fts: \n')
```

```
## Object class of mds_fts:
```

```
class(mds_fts)
```

```
## [1] "matrix" "array"
```

```
cat('Dimensions of mds_fts: \n')
```

```
## Dimensions of mds_fts:
```

```
dim(mds_fts)
```

```
## [1] 16763    50
```

```
# Responses
cat('Object class of resp_cc: \n')
```

```
## Object class of resp_cc:
```

```
class(resp_cc)
```

```
## [1] "integer"
```

```
cat('Length of resp_cc: \n')
```

```
## Length of resp_cc:
```

```
length(resp_cc)
```

```
## [1] 16763
```

```
# CPS Proficiency
cat('Object class of PVCPRO: \n')
```

```
## Object class of PVCPRO:
```

```
class(PVCPRO)
```

```
## [1] "numeric"
```

```
cat('Length of PVCPRO: \n')
```

```
## Length of PVCPRO:
```

```r
length(PVCPRO)
```

```
## [1] 16763
```

```r
cat('Summary statistics: \n')
```

```
## Summary statistics:
```

```r
summary(PVCPRO)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   76.08  433.03  504.13  499.22  569.55  828.46
```

We randomly select 80% of individuals for training the model (including cross-validation). The remaining 20% of individuals, i.e. the test set, is used for evaluating the prediction accuracy.

Predicting Overall Proficiency: The following code implements the prediction of overall CPS proficiency using ridge regression. Specifically, it performs of following steps:

- Randomly split the data into training (`index_train`) and testing (`index_test`) sets; Only the individuals in `index_train` are included for training the model.
- Getting the predictor features (`X`, 50 MDS features + one response accuracy feature)
- Ridge regression: The `cv.glmnet` function performs (by default, 10-fold) cross-validation. Arguments:
  - `x`: Matrix of predictors. Here, we use the mds features plus the final score
  - `y`: Predicted variable. Here, it is the overall CPS proficiency.
  - `family`: The model. Here, `gaussian` indicates linear model with continuous outcome.
  - `alpha`: Elastic net mixing parameter. Setting `alpha = 0` performs ridge regression with L_2 penalty.
- Predict individual PVCPRO using using the model returned above.
- Evaluate the out-of-sample correlation between the predictions and the observed values of Y on `index_test` individuals.

```r
library(glmnet)

set.seed(0000)

# splitting the data
samples <- which(!is.na(PVCPRO))
index_train <- sample(samples, .8*length(samples))
index_test <- setdiff(samples, index_train)

# getting the predictors
X <- cbind(mds_fts, resp_cc)

# MDS + response
out_mds <- cv.glmnet(x = X[index_train,], y = PVCPRO[index_train], family = "gaussian", alpha = 0)
PVCPRO_pred_mds <- predict(out_mds, newx = X, s = "lambda.min")


# Evaluate on test set
cat('Out-of-sample correlation w/ observed Ys: \n')
```

```
## Out-of-sample correlation w/ observed Ys:
```

```r
cor(PVCPRO_pred_mds[index_test,], PVCPRO[index_test])
```

```
## [1] 0.6638176
```

Let us compare the prediction accuracy with using responses alone.

```r
set.seed(0000)

# splitting the data
samples <- which(!is.na(PVCPRO))
index_train <- sample(samples, .8*length(samples))
index_test <- setdiff(samples, index_train)

# Response
out_resp <- lm(PVCPRO ~ resp_cc, subset = index_train)
PVCPRO_pred_resp <- predict(out_resp, newdata = as.data.frame(resp_cc))

# Evaluate on test set
cat('Out-of-sample correlation w/ observed Ys: \n')
```

```
## Out-of-sample correlation w/ observed Ys:
```

```r
cor(PVCPRO_pred_resp[index_test], PVCPRO[index_test])
```

```
## [1] 0.5788362
```

It can be observed that, the prediction of proficiency using the additional information from the processes (w/ OSR of .64) is more accurate than that using responses alone (OSR of .56). In other words, the processes can provide additional information on CPS proficiency.

Predicting Final Response: Prediction of final score on the climate control item (dichotomous) is similar, with just a few modifications. The code below implements score prediction using logistic regression with L_2 regularization. It performs of following steps (differences w/ continuous case are **bolded**):

- Randomly split the data into training (`index_train`) and testing (`index_test`) sets; Only the individuals in `index_train` are included for training the model.
- Getting the predictor features (`X`, 50 MDS features)
- Ridge regression: The `cv.glmnet` function performs (by default, 10-fold) cross-validation. Arguments:
  - `x`: Matrix of predictors. Here, we use the mds features
  - `y`: Predicted variable. Here, it is the participant score on the climate control item (0/1).
  - `family`: **The model. Here, `binomial` indicates logistic model with binary outcome.**
  - `alpha`: Elastic net mixing parameter. Setting `alpha = 0` performs ridge regression with L_2 penalty.
- Predict individual score using using the model returned above.
- Evaluate the agreement rate between the predictions and the observed values of Y on `index_test` individuals. Note that the predictions (`score_pred_mds`) is the prediction on the linear component of the logistic regression. To get dichotomous predictions, we can cut it at 0.

```r
library(glmnet)

set.seed(0000)

# splitting the data
samples <- which(!is.na(resp_cc))
```

```r
index_train <- sample(samples, .8*length(samples))
index_test <- setdiff(samples, index_train)

# getting the predictors
X <- mds_fts

# MDS
out_mds <- cv.glmnet(x = X[index_train,], y = resp_cc[index_train], family = "binomial", alpha = 0)
score_pred_mds <- predict(out_mds, newx = X, s = "lambda.min")
# dichotomize predictions
score_pred_mds <- 1*(score_pred_mds >= 0)

# Evaluate on test set
cat('Out-of-sample agreement w/ observed Ys: \n')
```

```
## Out-of-sample agreement w/ observed Ys:
```

```r
mean(score_pred_mds[index_test,] ==  resp_cc[index_test])
```

```
## [1] 0.7971965
```

Note that the baseline of prediction accuracy of is approximately .53 (53% of individuals in the test set responded correctly).

Application 2: Scoring

Our second application concerns the use of process features to increase measurement precision.

Accurate assessment of latent constructs in the key goal of a test. Suppose a test is designed to measure a unidimensional latent construct, θ. There are J items in the test. And for each item, both the problem-solving processes and the final response are available. There are many occasions in which the problem-solving processes can contain additional information on θ. For example:

- Partial completion: Some correct steps, but incorrect final response
- Strategy: Some strategies may suggest higher proficiency than others

For each item, the additional information from the processes about θ, when wisely used, increases amount of information an item can provide. This can potentially help us achieve two purposes:

- Increase measurement precision: With same number of items, additional information from the processes can increase measurement reliability.
- Reduce test length: At a preset level of reliability, additional information from processes can reduce the number of items required.

Method: Rao-Blackwellization Scheme

We introduce an approach to increase measurement accuracy using the process features. Consider a setting as follows:

- A test of J items is administered to a pretest sample of N individuals.
- For each individual i and each item j, the problem-solving processes and the final responses are both available.
- For each item, we extract process features using some feature extraction method (e.g., MDS). The extracted features (X_j) contain full information about the final response (Y_j). In other words, the X_j can perfectly predict Y_j.
- The test is originally scored based on final responses (Y_js) using an IRT model (e.g., the 2PL model or the graded response model).
  - Denote the IRT response-based ability estimate by $\hat{\theta}$;

– Suppose the item parameters (ζ_js) have been pre-calibrated on the pre-test sample.

Under this setting, the following procedures refine the latent trait estimate based on final responses, using the additional information on the problem-solving processes **leaving out item** j:

- Step 1: **Extracting a leave-one-out (LOO) sufficient statistic of** θ: This can be done by estimating the latent trait based on the final **response to item** j **only**, denoted $\hat{\theta}_{Y_j}$, and regressing it against **all process features except on item** j, denoted $\mathbf{X}_{-j}$. Call this regression outcome (LOO sufficient statistic) $T_{X_{-j}}$.
- Step 2: **Rao-Blackwellization**: With the output from the last step, $T_{X_{-j}}$, regress the response-based $\hat{\theta}$ **on all items** against $T_{X_{-j}}$ and Y_j. Call the regression output $\hat{\theta}_{-j}$.

Intuitively, the two steps performs two separate regressions:

- The first step estimates $T_{X_{-j}} = E(\hat{\theta}_{Y_j} \mid X_{-j})$.
- The second step estimates $\hat{\theta}_{-j} = E(\hat{\theta} \mid T_{X_{-j}}, Y_j)$.

Under some regularity assumptions, it can be shown theoretically that:

- $T_{X_{-j}}$, the output from Step 1, combined with Y_j, are sufficient statistics about θ.
- Based on the Rao-Blackwell theorem, regressing the original ability estimator ($\hat{\theta}$) on the sufficient statistic will result in a "better" ability estimator - one with lower mean-squared-error (MSE).

In other words, the LOO process-based latent trait estimate, $\hat{\theta}_{X_{-j}}$, will result in lower expected deviation from the true θ compared to the response-based latent trait estimate, $\hat{\theta}$. This holds for all j and all possible θs.

We can repeat this procedure and obtain the process-based $\hat{\theta}_{X_{-j}}$ for each j. To combine all J estimators, we can take the arithmetic average:

$$\hat{\theta}_X = \frac{1}{J}\sum_{j=1}^{J} \hat{\theta}_{X_{-j}}.$$

Illustration: Simulated data set

We illustrate the procedures of the above approach on a simulated data set. The full data set, available in `partial_example.RData`, consists of $J = 14$ items and $N = 2000$ examinees. It contains several objects:

- Problem-solving processes (`seqs_byitem`): A list (length-J), which contains the sequence of actions of each individual on each item. The sequences were simulated to be correlated with an underlying latent trait, true θ (object `theta`). Here's a preview for item 1, examinees 1 and 2:

```
seqs_byitem[[1]]$action_seqs[1:2]
```

```
## [[1]]
## [1] "a" "f" "f" "d" "e" "e" "g"
##
## [[2]]
## [1] "a" "f" "g"
```

- Final responses (`resp`): A matrix ($N \times J$), which contains the dichotomous responses of individuals on the J items. These responses were generated based on the presence/absence of the most correlated subsequence with θ.

```
head(resp)
```

```
##      item1 item2 item3 item4 item5 item6 item7 item8 item9 item10 item11 item12
## [1,]     1     0     1     1     0     0     0     1     1      1      0      0
## [2,]     0     0     0     0     0     0     0     0     0      0      0      0
```

```
## [3,]     1     0     0     1     0     1     1     1     0      1      1      1
## [4,]     0     1     0     0     1     0     1     1     0      1      1      1
## [5,]     0     0     0     0     1     1     0     1     1      0      0      0
## [6,]     0     0     0     0     0     0     0     0     0      0      0      0
##      item13 item14
## [1,]      1      1
## [2,]      0      0
## [3,]      1      1
## [4,]      1      1
## [5,]      1      0
## [6,]      0      0
```

In practice, the first step would be extracting features from the action sequences on each item. This can be done with the code below:

```
mds_fts_byitem <- lapply(seqs_byitem, seq2feature_mds, K = 10)
mds_fts_byitem <- lapply(mds_fts_byitem, function(x) x$theta)
# include resp as last dimension to ensure predictability
mds_fts_byitem <- Map(cbind, mds_fts_byitem, as.data.frame(resp))
```

This is going to take a while, so here, the pre-trained 10−dimensional MDS features for each item is stored as `mds_fts_byitem`. This is a length-J list, where each element is a 2000×11 matrix, the last dimension being the final response to item j.

For the sake of illustration, let's only use the **first** 5 **items** out of all 14. In this case, the ability estimate based on traditional IRT models, $\hat{\theta}$, will depend only on the dichotomous responses to 5 questions.

```
resp_ps <- resp[,1:5]
mds_fts_ps <- mds_fts_byitem[1:5]
```

Using the process information, we will produce a process-based latent trait estimate ($\hat{\theta}_X$). Let's try to answer the following questions:

- How do the response-based and process-based ability estimators perform, when the test consists of only 5 questions?
- Comparing the two estimators - what does the process-based ability estimator do differently?

Obtaining both estimators require the installation of the `mirt` package (`install.packages("mirt")`). Let's define two simple functions for IRT-based item calibration and ability estimation:

- `get_model_mirt`: This fits the IRT model and calibrates the item parameters based on final responses.
- `get_traits_mirt`: This obtains the latent trait estimate based on the responses to a subset of items. We use EAP estimation in the illustrations, but this can be changed to other methods.

```
get_model_mirt <- function(scores, types, itemset){
  model <- mirt(data = scores[,itemset], model = 1, itemtype = types[itemset],verbose = F)
  return(model)
}

# get theta based on subset of item responses with known irt pars
get_traits_mirt<-function(scores, model, itemset, method = 'EAP'){
  scores[,which(! colnames(scores) %in% itemset)] <- NA
  res <- fscores(model, response.pattern = as.matrix(scores), method = method, full.scores.SE = TRUE)
  theta <- res[,'F1']
  if(method == 'ML'){
    theta[which(theta == Inf)] <- max(theta[which(theta<Inf)])+.5
    theta[which(theta == -Inf)] <- min(theta[which(theta>-Inf)])-.5
  }
```

```
  return(theta)
}
```

The following code implements the partial scoring procedures. Again, we train the partial scoring model on a training set (`index_train`) and set aside a test set (`index_test`) for evaluations.

```
library(glmnet)
library(mirt)

item_code <- colnames(resp_ps)
item_type <-  ifelse(apply(resp_ps, 2, function(x)
   length(unique(x)))==2, '2PL','graded')

set.seed(0000)
samples <- 1:nrow(resp_ps)
index_train <- sample(samples, .8*length(samples))
index_test <- setdiff(samples, index_train)

# Calibrate IRT model parameters
mirt_model <- get_model_mirt(resp_ps[index_train,],item_type, itemset = item_code)
# response-based trait estimate
theta_yhat <- get_traits_mirt(resp_ps, mirt_model, item_code)

# obtain LOO process-based score for each j
thetahats_loo <- matrix(NA, nrow(resp_ps), ncol(resp_ps))
for(j in 1:ncol(resp_ps)){
  # get theta_yj based on Yj
  theta_yj <- get_traits_mirt(resp_ps, mirt_model, item_code[j])
  # get T_-j: Regress theta_yj on X[-j]
  features <- Reduce(cbind, mds_fts_ps[-j])              # all process fts except item j
  reg_X <- cv.glmnet(x = features[index_train,], y = theta_yj[index_train], family = "gaussian", alpha =
  Tx <- predict(reg_X, newx = features, s = "lambda.min")    # T_-j
  # get thetahat_X,-j: Regress theta_yhat on Yj, T_-j
  df <- data.frame(theta_yhat, Yj = resp_ps[,j], Tx = Tx)
  reg_T <- lm(theta_yhat ~ Yj+Tx+Yj*Tx,data = df, subset = index_train)
  thetahats_loo[,j] <- predict(reg_T, newdata = df)
}

theta_x <- rowMeans(thetahats_loo)
```

The true θ used for generating the sequences and responses is stored as `theta_ps`. On the test set, we can evaluate the response- and process-based latent trait estimates in terms of two aspects:

- MSE with respect to true θ
- Kendall's τ correlation with true θ

```
# Evaluation:

# MSE
cat('MSE of process-based theta estimate: \n')
```

```
## MSE of process-based theta estimate:
```

```
mean((theta_x[index_test] - theta_ps[index_test])^2)
```

```
## [1] 0.3314686
```

```
cat('MSE of response-based theta estimate: \n')
```

```
## MSE of response-based theta estimate:
```

```
mean((theta_yhat[index_test] - theta_ps[index_test])^2)
```

```
## [1] 0.3907637
```

```
# Correlation (Kendall's tau)
cat('Correlation using process-based theta estimate: \n')
```

```
## Correlation using process-based theta estimate:
```

```
cor(theta_x[index_test], theta_ps[index_test], method = 'kendall')
```

```
## [1] 0.644411
```

```
cat('Correlation using response-based theta estimate: \n')
```

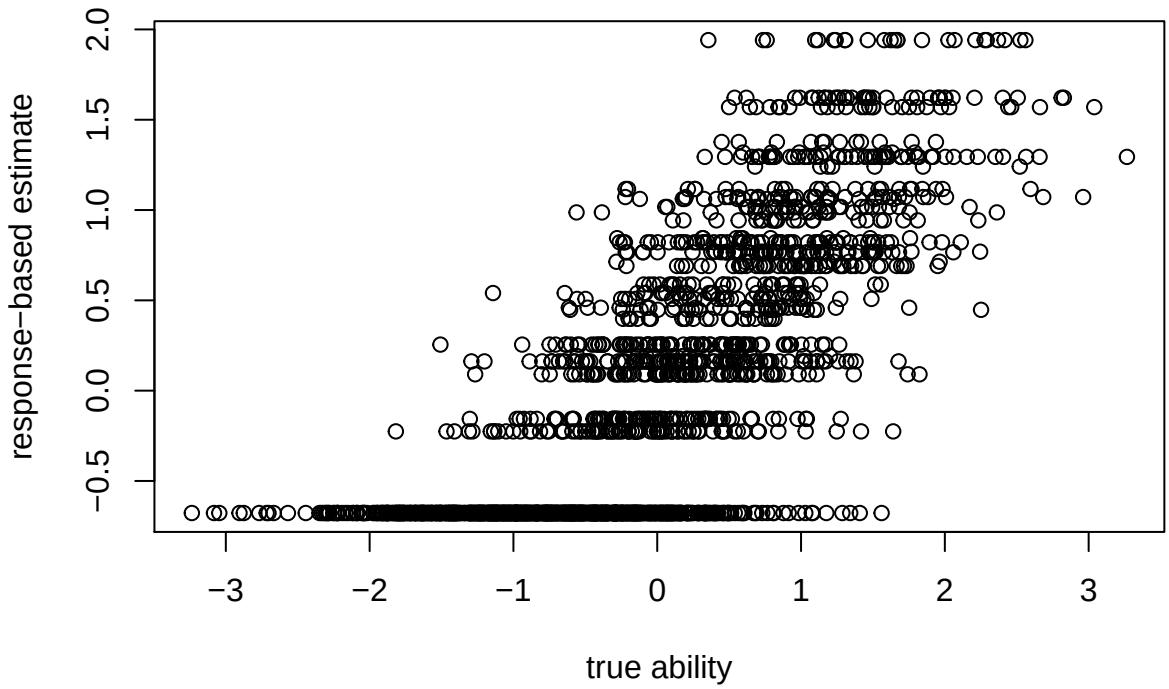
```
## Correlation using response-based theta estimate:
```

```
cor(theta_yhat[index_test], theta_ps[index_test], method = 'kendall')
```

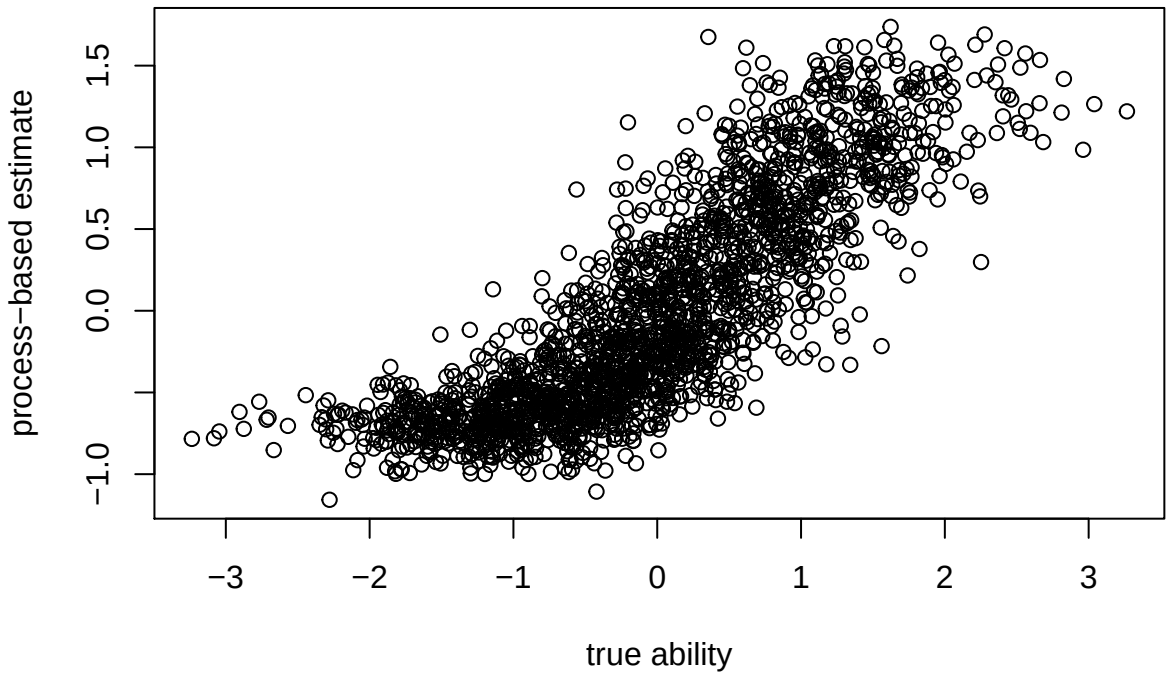
```
## [1] 0.6153846
```

Why does the process-based estimator agree better with the true θ? Let's take a look at the scatterplots of the two estimators against true θ.

```
# Plots:
plot(theta_ps, theta_yhat, xlab = 'true ability', ylab = 'response-based estimate')
```



```r
plot(theta_ps, theta_x, xlab = 'true ability', ylab = 'process-based estimate')
```



Application 3: DIF Correction

Dimensional Account of DIF

Differential item functioning (DIF) occurs when an item functions differently for separate groups of test-takers. Under a dimensional account, DIF can be explained by the presence of construct-irrelevant, or "nuisance" traits, that affect response accuracy. When the distribution of the nuisance trait differs across two groups (i.e., the **focal** (f) and the **reference** (r) groups), one group of individuals will have a lower chance of responding correctly. For example:

- For a statistics assessment: A question involving baseball rules may be harder for test-takers from cultures with little baseball exposure. Here, the nuisance trait is baseball knowledge.
- For a computer-based assessment: A question presented with non-adjustable small font size may be harder for test-takers with low-vision. Here, the nuisance trait is the ability to read and work with small characters.

When these nuisance traits are not taken into account in item response modeling, the item parameters will differ for individuals from the two groups. Consequently, the item will be biased against some of individuals, threatening test validity.

DIF Removal with Process Features

For a traditional assessment with response information only, most of the time, the best one can do is to identify items with DIF and remove them from the test. However, additional problem-solving process information may allow us to reconstruct the **nuisance variable** leading to DIF. By finding the nuisance variable, we

may **reconstruct the "true" item response function** that depends on **both measured and nuisance constructs**, thus correcting for DIF in the measurement model.

Essentially, we want to find the nuisance trait (η_j) from the process features, so that group membership no longer affects the item response given (θ, η). Taking the 2PL model as an example. For an item (j) with detected DIF (for example, using a likelihood ratio test), we want to find a linear combination of the process features,

$$\eta_j = \sum_{k=1}^{K} \omega_{jk} X_{jk},$$

so that the probability of correct response in the focal group, $P_f(Y_j = 1 \mid \eta_j, \theta)$, and the correct response probability in the reference group, $P_r(Y_j = 1 \mid \eta_j, \theta)$, are as close as possible. Here, the probability of correct response (IRF) given group membership g is

$$P_g(Y_j = 1 \mid \theta, \eta_j) = \frac{1}{1 + \exp[-(a_{gj}(\theta + \eta_j) + d_{gj})]}.$$

Note that there is not an unique η that achieves this. For example, if all K process features are used, the final response is perfectly predictable. This (perfect) response function will be group invariant, but, at the same time, $\theta-$invariant, meaning that the response to this question will no longer provide information for estimating θ. Thus we want to perform **variable selection** and select a **subset of process features** to reconstruct the nuisance trait η.

Here we introduce a method to correct for DIF by

(1) Identifying items with DIF using likelihood ratio test
(2) For items with DIF, reconstruct nuisance trait η with a subset of process features. This is done so by estimating ωs to minimize the L_2 distance between focal and reference group item response functions given $\tilde{\theta} = \theta + \eta_j$, i.e.,

$$d_j = \sqrt{\int [P_f(Y_j = 1 \mid \tilde{\theta}) - P_r(Y_j = 1 \mid \tilde{\theta})]^2 f(\tilde{\theta}) d\tilde{\theta}}.$$

(3) Using the reconstructed nuisance traits of DIF items, η_js, and the updated estimates of a_j and d_js, update the IRF for items with DIF to incorporate η_j, and re-estimate θ.

The test statistic d_j, which is empirically estimated by $\hat{d}_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[P_f(Y_j = 1 \mid \tilde{\theta}) - P_r(Y_j = 1 \mid \tilde{\theta})]^2)}$, also allows us to perform **hypothesis testing** to see whether the two IRFs for focal and reference groups significantly differ. Under the Null hypothesis that two IRFs given θ and η are identical, i.e.,

$$H_0 : a_{jf} = a_{jr} = a_{j0}; d_{jf} = d_{jr} = d_{j0}, \boldsymbol{\omega}_{jf} = \boldsymbol{\omega}_{jr} = \boldsymbol{\omega}_{j0},$$

the L_2 distance test statistic, $\hat{d}_j$, has asymptotic generalized χ^2 distribution. The null distribution of $\hat{d}_j$ can also be simulated, allowing us to test whether the IRFs significantly differ across groups.

Illustration

Here, we use a simulated data set to illustrate the DIF correction procedures. The dataset for the current example is available in `DIF_example.RData`. The response data ($J = 15$, `responses_dif`) is simulated so that the first item has DIF against the focal group (`group == 'F'`). The correct response to item 1 depends on both θ (`theta`) and a nuisance variable (`nuisance`).

What we want to do is to use the process features of item 1 (`fts_item_1`), which contain both θ and nuisance trait information, to reconstruct the nuisance variable (η), so that DIF of this item can be corrected by incorporating η into the measurement model.

First let us look at some summary statistics:

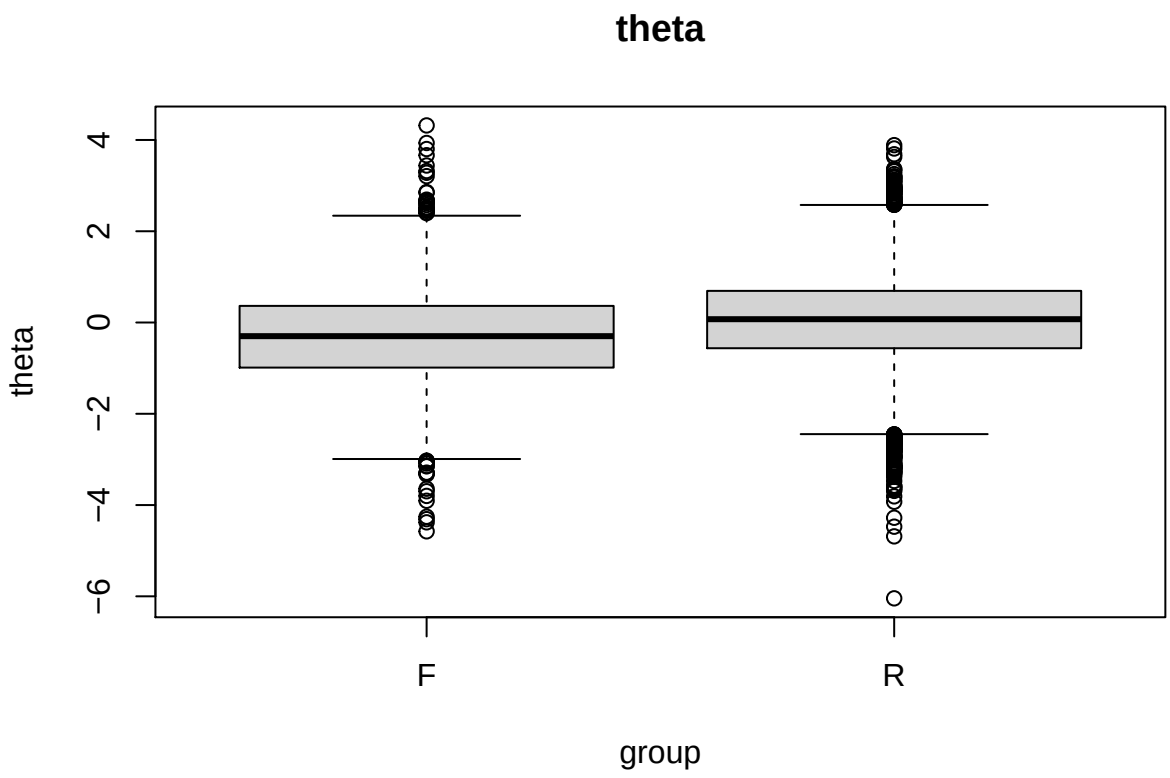
- Frequency of focal and reference group membership

- Distribution of true θ in these two groups
- Distribution of true nuisance η in these two groups

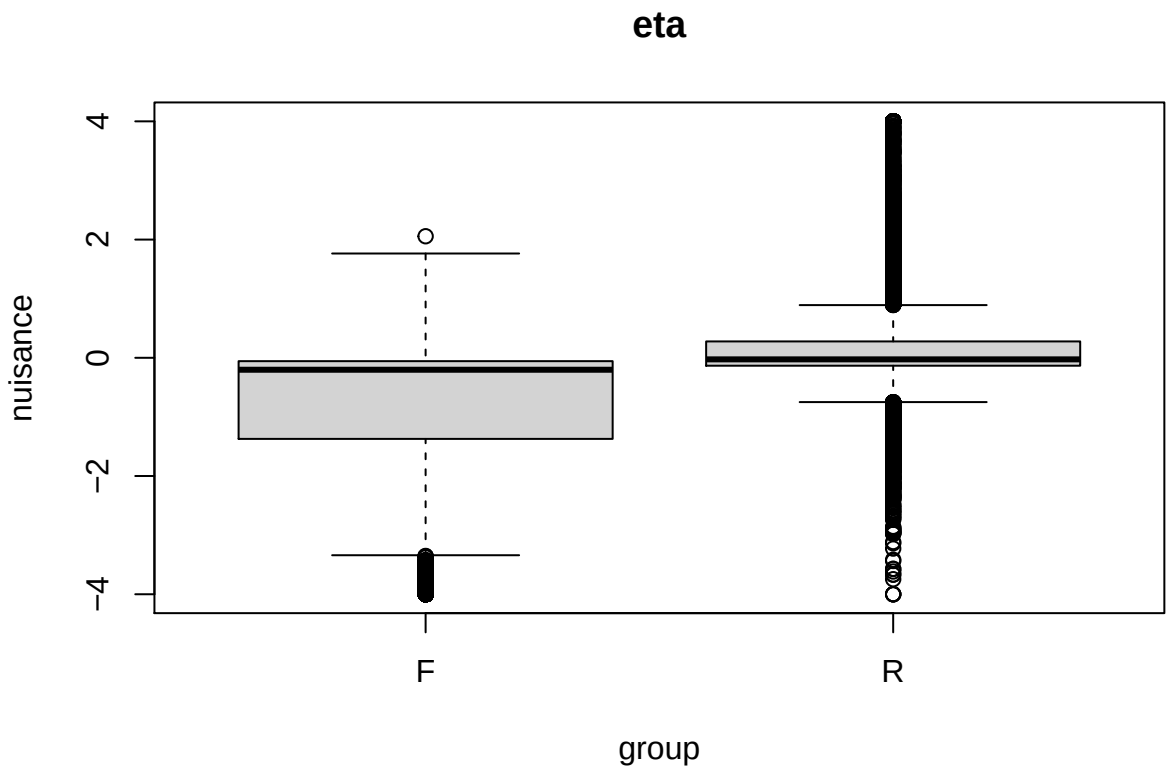
```
table(group)
```

```
## group
##     F     R
##  2429 14334
```

```
boxplot(theta ~ group, main = 'theta')
```




```
boxplot(nuisance ~ group, main = 'eta')
```



Next, we use the `mirt` package to perform likelihood ratio test to determine whether the first 3 items have DIF. We use items 4 to 15 as the anchor set, i.e., items assumed to have no DIF.

```
library(mirt)
dif_model <- multipleGroup(responses_dif, 1, group = group,
                           invariance = c(colnames(responses_dif)[-(1:3)],'free_means','free_var')
                           )
```

```
## Iteration: 1, Log-Lik: -128752.146, Max-Change: 0.93743Iteration: 2, Log-Lik: -126758.595, Max-Chang
```

```
DIF(dif_model, c('a', 'd'), items2test = 1:3)
```

```
##             AIC     AICc    SABIC       HQ      BIC      X2 df     p
## item_1 -229.521 -229.512 -224.972 -226.971 -221.794 231.521  1 0.000
## item_2   -0.142   -0.132    4.407    2.408    7.585   2.142  1 0.143
## item_3    0.477    0.486    5.026    3.027    8.204   1.523  1 0.217
```

It can be seen that only item 1 was found to have significant DIF. In what follows, we try to correct for the DIF in item 1 using process features from this item. This is done through forward selection that minimizes the L_2 difference in IRFs of the two groups. Specifically, the following steps are performed:

- On the anchor set (items without DIF, here, items 2 to 15), get an initial estimate the θ, $\hat{\theta}_0$.
- Initialize model: fit 2PL (logistic regression) with one predictor, $\hat{\theta}_0$, separately for focal and reference groups. Calculate $\hat{d}_j$ and calculate the $p-$value for testing H_0. Here, we expect H_0 to be rejected because the IRF should be different for the two groups, when conditioned on θ only. Use this as the *base model*.
- For $k = 1 : K$, with K as the maximum number of process features to include for η (K, less than or equal to the number of process features), iterate the following:
  - For each of the features that are not yet included in the base model, add the feature to the base

model. Refit the logistic model for focal and reference groups, calculate $\hat{d}_j$ for the model with the new candidate feature.
  - The model that (1) has significant θ and new feature coefficients and (2) achieves the lowest $\hat{d}_j$ will become the *updated base model*. The newly added feature in this model is added to the set of features for reconstructing η.
- Select the model with k^* new features, where k^* is the dimension $(1:K)$ that achieves lowest $\hat{d}_j$, as the final model. The first k^* selected process features are incorporated for η reconstruction. Denote the selected set of features by $\mathcal{K}_j$.
- Test the hypothsis H_0 on the final model to verify that DIF has been removed.
- On full data (focal + reference), fit the logistic regression model, $P(Y_j = 1 \mid \theta_0, \mathbf{X}_{\mathcal{K}_j}) = \frac{1}{1+\exp(-[\beta_0\theta_0+\sum_{k\in\mathcal{K}_j}\beta_k X_{jk}+\delta])}$. The new item parameters for the IRF $P_g(Y_j \mid \theta, \eta_j)$ will be:
  - $a_j = \beta_0$
  - $d_j = \delta$
  - $\omega_{jk} = 0$ for k not in $\mathcal{K}_j$, and $\omega_{jk} = \frac{\beta_k}{\beta_0}$ for $k \in \mathcal{K}_j$.

To implements this. We make use of two functions:

- `DIF_forward`: This function implements forward selection of process features to minimize the L_2 distance between the two groups' $P(Y_j = 1 \mid \theta_0, \eta_j = \sum_{k=1}\omega_{jk}X_{jk})$. The arguments are as follows:
  - `features`: The matrix of process features ($N \times K$) for the item with DIF
  - `response`: Vector of response (length N) for the item with DIF
  - `group`: Vector of group membership (`'F'` or `'R'`)
  - `theta_ref`: The initial θ estimate from anchor set, $\hat{\theta}_0$
  - `K`: Max number of process features to be selected. Default is the dimension of `features`.

This function will return the order of feature selection, the L_2 distance between the two groups' IRFs using these features, and the $p-$value for testing H_0.

```
DIF_forward <- function(features,          # process features from DIF item
                        response,          # response vector on the DIF item
                        group,             # vector of group membership, 'F' or 'R'
                        theta_ref,         # latent trait estimate on the anchor set (items w/o DIF)
                        K = NULL           # max number of feature dimensions to add (default to featur
){
  # reorganize data
  Xs <- cbind(theta_ref, features)
  Xf <- Xs[which(group == 'F'),]
  Xr <- Xs[which(group == 'R'),]
  Yf <- response[which(group == 'F')]
  Yr <- response[which(group == 'R')]

  if( is.null(K)) K <- ncol(features)
  results_by_k <- matrix(NA, (K+1), 2)
  colnames(results_by_k) <- c('L_2', 'p')

  # initialize --- theta only in irt model

  Xf_tmp <- as.matrix(Xf[,1])
  Xr_tmp <- as.matrix(Xr[,1])
  results_by_k[1,] <- DIF_p_value_sim(Xf_tmp, Xr_tmp, Yf, Yr, sim = T)
  cat('---   Initial model    ------ \n  L2 dist: ',results_by_k[1, 1],
      '\n  p-value: ',results_by_k[1,2],'\n')

  # add process features one by one, each feature chosen to minimize l2 distance
  add_index <- c()
```

```
  for(dim in 1:K){
    remain_index <- setdiff((1:ncol(features)), add_index)

    # calculate l_2 for each added candidate feature
    p_value_temp <- foreach(k = 1:length(remain_index), .combine = c, .noexport = c('matmult_cpp', 'runi
      # already selected
      Xs_tmp <- as.matrix(Xs[,c(1, (add_index+1))])
      # add new candidate feature
      Xs_tmp <- cbind(Xs_tmp, Xs[,(remain_index[k]+1)])
      # split to focal and reference
      X_focal <- Xs_tmp[which(group == 'F'),]
      X_ref <- Xs_tmp[which(group == 'R'),]
        tryCatch(
        DIF_p_value_sim(Xf = X_focal, Xr = X_ref, Yf = Yf, Yr = Yr, sim = F),
        error = function(e) c(1, 0)
      )
    }
    p_value_temp <- matrix(p_value_temp, nrow = 2)

    stopifnot(dim(p_value_temp) == c(2, length(remain_index)))
    if(sum(p_value_temp[1,]) == length(remain_index)){
      break
    }
    # select feature with lowest L_2
    k <- which.min(p_value_temp[1,])
    add_index <- c(add_index, remain_index[k])

    # get p value
    Xs_tmp <- Xs[,c(1,(add_index+1))]
    # split to focal and reference
    X_focal <- Xs_tmp[which(group == 'F'),]
    X_ref <- Xs_tmp[which(group == 'R'),]
    results_by_k[dim+1,] <- tryCatch(
      DIF_p_value_sim(Xf = X_focal, Xr = X_ref, Yf = Yf, Yr = Yr, sim = T),
      error = function(e) c(1, 0, 0, 0, 0, 0)
    )


    cat('--- K =',dim,'Adding ft', remain_index[k],'------ \n  L2 dist: ',results_by_k[(dim+1), 1],
        '\n  p-value: ',results_by_k[(dim+1),2],'\n')

  }

  results_by_k <- cbind(results_by_k, c(NA, add_index))
  colnames(results_by_k)[3] <- 'Added_ft'
  return(results_by_k)

}
```

- `get_itempar_DIF`: Using the output from `DIF_forward`, get the item parameters $(a_j, d_j, \boldsymbol{\omega}_j)$ for the final model. Arguments are as follows:
  - `features`: The matrix of process features ($N \times K$) for the item with DIF
  - `response`: Vector of response (length N) for the item with DIF

- `theta_ref`: The initial θ estimate from anchor set, $\hat{\theta}_0$
- `DIF_forward_result`: The output from `DIF_forward`
- `K`: Final number of process features to be incorporated. Default is the k achieving the lowest L_2 distance.

```
# Updated IRT model w/ Nuisance
get_itempar_DIF <- function(features,            # process features from DIF item
                            response,            # response vector on the DIF item
                            theta_ref,           # latent trait estimate on anchor set (items w/o DIF)
                            DIF_forward_result,  # output matrix from DIF_forward (index of selected feat
                            K = NULL             # feature dimension to add (default to dimension with sm
){
  # Default K: lowest L2 distance between focal and reference irfs
  if(is.null(K)){
    K <- which.min(DIF_forward_result[,'L_2'])-1
  }

  # get feature set
  add_index <- DIF_forward_result[1:K+1,'Added_ft']

  # fit logistic
  Xs <- cbind(1, theta_ref, features[,add_index])
  pars <- glm.fit(Xs, response, family = binomial(), intercept = F)$coefficients
  d <- pars[1]
  a <- pars[2]
  omegas <- numeric(ncol(features))
  omegas[add_index] <- pars[-(1:2)]/a

  return(list(a = a, d = d, omegas = omegas))
}
```

Because the variable selection can take a few minutes when K is large, we set $K = 10$ in the current illustration.

```
item_DIF <- 1
init_model <- mirt(responses_dif[,-item_DIF], 1, '2PL')
```

```
## Iteration: 1, Log-Lik: -118814.750, Max-Change: 0.52620Iteration: 2, Log-Lik: -117332.997, Max-Chang
```

```
theta_init <- fscores(init_model)

DIF_results <- DIF_forward(fts_item_1,
                           responses_dif[,1],
                           group,
                           theta_init,
                           K = 10)
```

```
## ---   Initial model    ------
##   L2 dist:  0.1949156
##   p-value:  0
## --- K = 1 Adding ft 16 ------
##   L2 dist:  0.08087156
##   p-value:  0
## --- K = 2 Adding ft 10 ------
##   L2 dist:  0.06391115
##   p-value:  0
```

```
## --- K = 3 Adding ft 4 ------
##   L2 dist:  0.03704745
##   p-value:  0.03
## --- K = 4 Adding ft 8 ------
##   L2 dist:  0.03393897
##   p-value:  0.06
## --- K = 5 Adding ft 14 ------
##   L2 dist:  0.03335343
##   p-value:  0.05
## --- K = 6 Adding ft 28 ------
##   L2 dist:  0.03378653
##   p-value:  0.09
## --- K = 7 Adding ft 9 ------
##   L2 dist:  0.03489171
##   p-value:  0.14
## --- K = 8 Adding ft 13 ------
##   L2 dist:  0.03782261
##   p-value:  0.17
## --- K = 9 Adding ft 15 ------
##   L2 dist:  0.0396233
##   p-value:  0.08
## --- K = 10 Adding ft 50 ------
##   L2 dist:  0.04547892
##   p-value:  0.05
```

```
itempar_dif_1 <- get_itempar_DIF(features = fts_item_1,
                                 response = responses_dif[,1],
                                 theta_ref = theta_init,
                                 DIF_forward_result = DIF_results)

as.data.frame(DIF_results)
```

```
##           L_2    p Added_ft
## 1  0.19491555 0.00       NA
## 2  0.08087156 0.00       16
## 3  0.06391115 0.00       10
## 4  0.03704745 0.03        4
## 5  0.03393897 0.06        8
## 6  0.03335343 0.05       14
## 7  0.03378653 0.09       28
## 8  0.03489171 0.14        9
## 9  0.03782261 0.17       13
## 10 0.03962330 0.08       15
## 11 0.04547892 0.05       50
```

```
itempar_dif_1
```

```
## $a
##        F1
## 0.6139196
##
## $d
##
## -1.371837
##
```

```
## $omegas
##  [1]  0.0000000  0.0000000  0.0000000  7.5018459  0.0000000  0.0000000
##  [7]  0.0000000  0.5439357  0.0000000  4.1179758  0.0000000  0.0000000
## [13]  0.0000000 -0.5982839  0.0000000  7.9552883  0.0000000  0.0000000
## [19]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [25]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [31]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [37]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [43]  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000
## [49]  0.0000000  0.0000000
```

The last step would be to re-estimate the target trait θ, with:

- the updated IRF for the DIF item, and
- the responses + original IRFs for the rest of the items.

This is done with the `update_theta` function. Arguments:

- `responses`: The response matrix ($N \times J$)
- `item_DIF`: Index of items exhibiting DIF
- `fts_DIF`: A list (length same as `item_DIF`) of DIF items' features. Each element is feature matrix of the corresponding DIF item
- `as_anchor`, `ds_anchor`: vectors of a and d parameters for items without DIF.
- `itempars_DIF`: list of DIF item parameters, each element is the output from `get_itempar_DIF` for the DIF item
- `method`: Method used to estimate θ. Default is `'EAP'`, which performs Bayesian expected a posteriori estimation with standard normal prior. Another option is `'MLE'` with $(-4, 4)$ as the lower and upper bounds.

```r
update_theta <- function(responses,              # response matrix
                         item_DIF,               # vector of DIF item index
                         fts_DIF,                # list of DIF item features, each element is ft mat of
                         as_anchor,              # vector of a parameters for items w/o DIF
                         ds_anchor,              # vector of d parameters for items w/o DIF
                         itempars_DIF,           # list of DIF item parameters, each element is output o
                         method = 'EAP'          # trait estimation method; 'EAP': N(0,1) prior; 'MLE':
){

  # get nuisance of each item (for no dif items this is 0)
  etamat <- matrix(0, nrow(responses), ncol(responses))
  for(i in 1:length(item_DIF)){
    fts <- fts_DIF[[i]]
    itempars <- itempars_DIF[[i]]
    etamat[,item_DIF[i]] <- fts %*% itempars$omegas
  }

  # get item slope and intercepts
  as <- ds <- rep(NA, ncol(responses))
  as[-item_DIF] <- as_anchor
  ds[-item_DIF] <- ds_anchor
  as[item_DIF] <- sapply(itempars_DIF, function(x) x$a)
  ds[item_DIF] <- sapply(itempars_DIF, function(x) x$d)

  # estimate theta
  if(method == 'EAP'){
    theta_pts <- seq(-4, 4, .2)
```

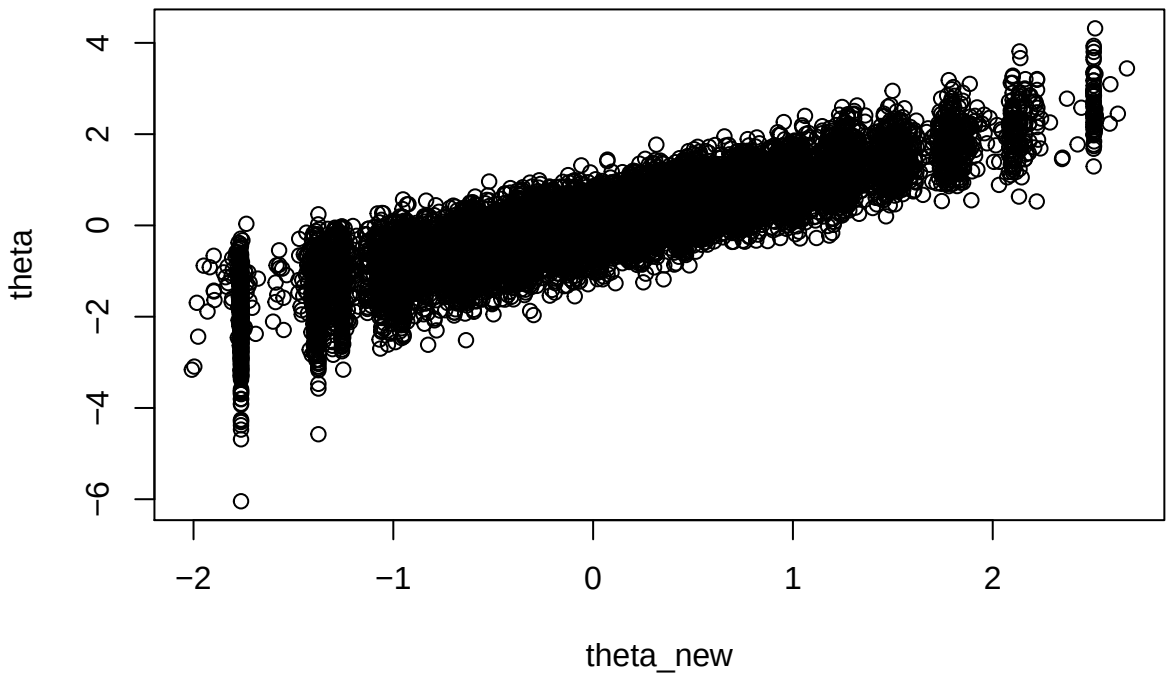
```
    theta_density <- rep(1, nrow(responses)) %o% (dnorm(theta_pts) * .2)
    like_theta <- matrix(NA, nrow(responses), length(theta_pts))
    for(t in 1:length(theta_pts)){
      theta <- theta_pts[t]

      # get likelihood | theta
      like <- rep(1, nrow(responses))
      for(i in 1:ncol(responses)){
        like <- like * (p_dif(theta, etamat[,i], as[i], ds[i])^responses[,i]) *
          ((1-p_dif(theta, etamat[,i], as[i], ds[i]))^(1-responses[,i]))
      }
      like_theta[,t] <- like
    }

    theta_est <- colSums(theta_pts * t(theta_density * like_theta))/(rowSums(theta_density * like_theta

  }
  if(method == 'MLE'){
    theta_est <- rep(NA, nrow(responses))
    for(n in 1:nrow(responses)){
      theta_est[n] <- optimise(ll_dif, lower = -4, upper = 4, as = as, ds = ds,
                               etas = etamat[n,], response = responses[n,])$minimum
    }
  }
  return(theta_est)
}
```

Here we plot the updated $\hat{\theta}$ (`theta_new`) against the true theta to evaluate its accuracy. We can also look at the relationship between true and reconstructed nuisance.

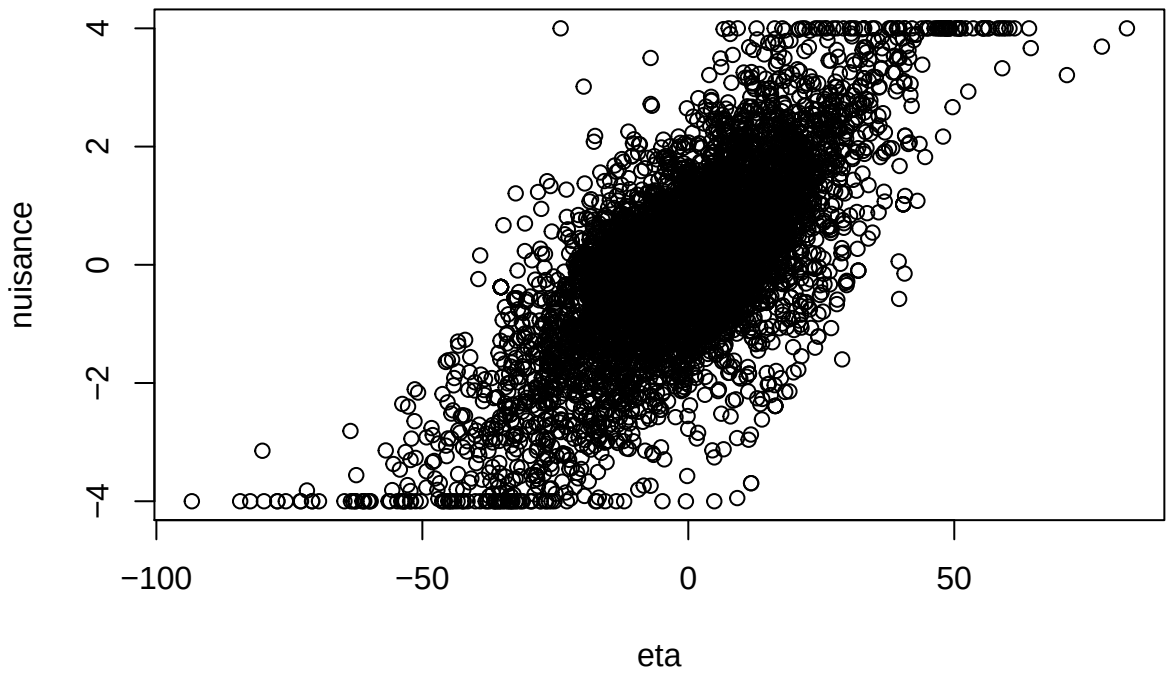
```
item_DIF <- 1
init_model <- mirt(responses_dif[,-item_DIF], 1, '2PL')
```

```
## Iteration: 1, Log-Lik: -118814.750, Max-Change: 0.52620Iteration: 2, Log-Lik: -117332.997, Max-Chang
```

```
# --- reestimate theta incorprating nuisance for dif items ---------------------------

fts_DIF <- list(fts_item_1)
itempars_DIF <- list(itempar_dif_1)
itempar_anchor <- coef(init_model, simplify = T)$items
as_anchor <- itempar_anchor[,'a1']
ds_anchor <- itempar_anchor[,'d']

theta_new <- update_theta(responses = responses_dif,
                          item_DIF = item_DIF,
                          fts_DIF = fts_DIF,
                          as_anchor = as_anchor,
                          ds_anchor = ds_anchor,
                          itempars_DIF = itempars_DIF,
                          method = 'EAP')

plot(theta_new, theta)
```




```
eta <- fts_item_1 %*% itempar_dif_1$omegas
plot(eta, nuisance)
```


nuisance
4
2
0
−2
−4
−100
−50
0
50
eta