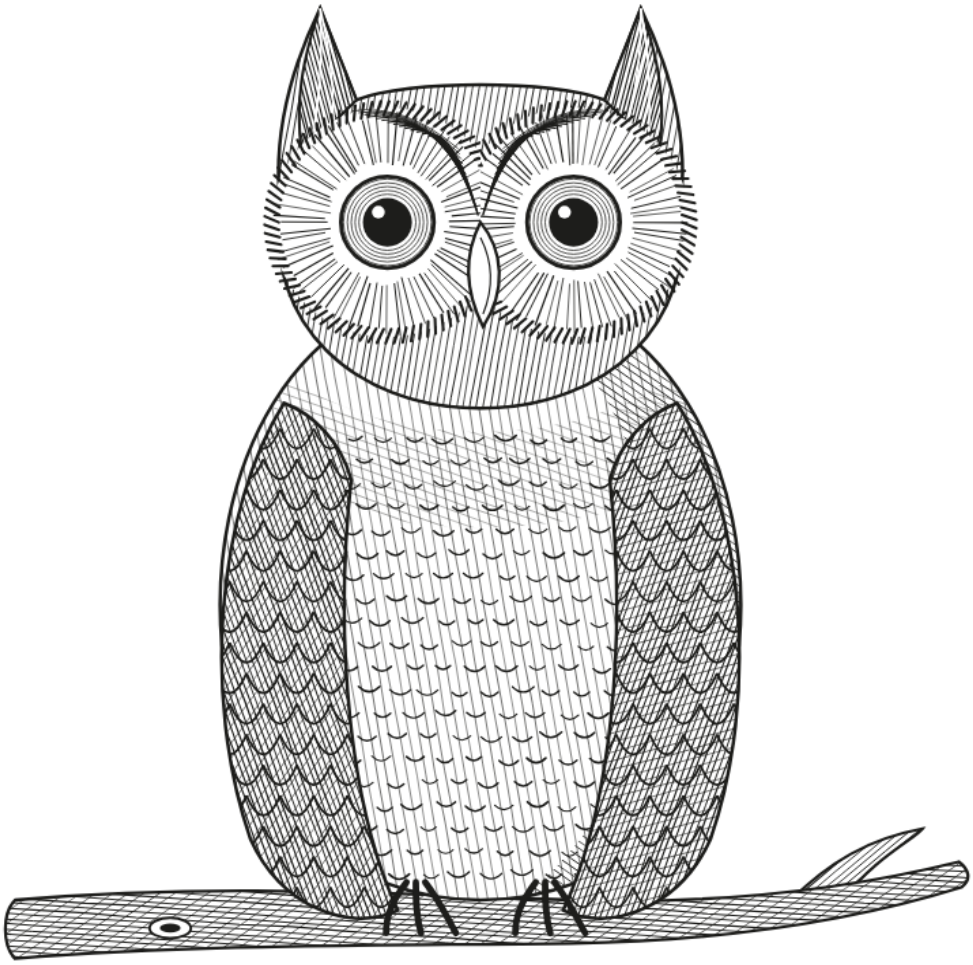
Notes for teaching and learning



Item Response Theory

with mirt and mirtverse

A hands-on course in measurement with R

mirtverse

Okan Bulut

Item Response Theory

with mirt and mirtverse

Notes for teaching and learning

Version 1.1

Okan Bulut

Measurement, Evaluation, and Data Science
University of Alberta

bulut@ualberta.ca · www.okanbulut.com

Item Response Theory with mirt and mirtverse: Notes for Teaching and Learning
by Okan Bulut

© 2026 Okan Bulut. The text and figures of this book are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International licence (CC BY-NC-SA 4.0). The R code is licensed under GPL-3.

Version 1.1
DOI: 10.5281/zenodo.23885605
Online edition: https://www.okanbulut.com/mirtverse/book/
Source: https://github.com/okanbulut/mirtverse/tree/main/book

Typeset with bookdown and XeLaTeX on November 12, 2026. The cover illustration is an original engraving-style drawing made for this book.

Contents

Preface **11**
How the book is organised . . . 11
Before you start . . . 13
Other teaching materials . . . 13
Using and adapting the book . . . 13
Other formats . . . 13
How to cite this book . . . 13
Licence . . . 14
Acknowledgements . . . 14

About the author **15**

1 Measurement, scoring and classical item analysis **17**
1.1 The ideas . . . 17
1.2 Example 1: A multiple-choice science test . . . 18
1.3 Example 2: A wellbeing questionnaire . . . 20
1.4 A first look with mirtverse . . . 22
1.5 Your turn . . . 22
In the lab . . . 23
Further reading . . . 23

2 The Rasch, 2PL and 3PL models **24**
2.1 The ideas . . . 24
2.2 Example 1: The LSAT section 7 items . . . 25
2.3 Example 2: A driving theory test with guessing . . . 29
2.4 Your turn . . . 30
In the lab . . . 31
Further reading . . . 31

3 Estimation and comparing models **32**
3.1 The ideas . . . 32
3.2 Example 1: A test the Rasch model fits . . . 33
3.3 Example 2: When the criteria disagree . . . 34
3.4 Your turn . . . 36
In the lab . . . 37
Further reading . . . 37

4 Dimensionality and local independence **38**
4.1 The ideas . . . 38
4.2 Example 1: A mathematics test with a reading dimension . . . 39
4.3 Example 2: A reading passage that ties four items together . . . 41
4.4 Your turn . . . 43
In the lab . . . 45
Further reading . . . 45

5 Item fit, person fit and item quality **46**
5.1 The ideas . . . 46
5.2 Example 1: The same items under two models . . . 47
5.3 Example 2: Finding aberrant response patterns . . . 50
5.4 Your turn . . . 52
In the lab . . . 53
Further reading . . . 53

6 Information, precision and reliability **54**
6.1 The ideas . . . 54
6.2 Example 1: Where does the science test measure well? . . . 55
6.3 Example 2: Designing a test for a decision . . . 57
6.4 Your turn . . . 59
In the lab . . . 60
Further reading . . . 60

7 Scoring people **61**
7.1 The ideas . . . 61
7.2 Example 1: Four estimators, one short test . . . 62
7.3 Example 2: A proficiency cut on a mathematics test . . . 63
7.4 Your turn . . . 66
In the lab . . . 67
Further reading . . . 67

8 Polytomous items and open-ended responses **68**
8.1 The ideas . . . 68
8.2 Example 1: Attitudes to science . . . 69
8.3 Example 2: A rating category nobody uses . . . 72
8.4 Open-ended responses and raters . . . 73
8.5 Your turn . . . 74
In the lab . . . 75
Further reading . . . 75

9 Differential item functioning and fairness **76**
9.1 The ideas . . . 76
9.2 Example 1: Paper versus tablet . . . 77
9.3 Example 2: Mantel-Haenszel on a chemistry quiz . . . 81
9.4 Your turn . . . 82
In the lab . . . 83
Further reading . . . 83

10 Linking and equating **84**
10.1 The ideas . . . 84
10.2 Example 1: Separate calibration and linking . . . 85
10.3 Example 2: Concurrent calibration and equating a pass mark . . . 88
10.4 Your turn . . . 90
In the lab . . . 91
Further reading . . . 91

11 Item banks, test assembly and adaptive testing **92**
11.1 The ideas . . . 92
11.2 Example 1: A vocabulary bank and a fixed form . . . 93
11.3 Example 2: A CAT from scratch . . . 95
11.4 Your turn . . . 97
In the lab . . . 98
Further reading . . . 98

12 Cut scores, reporting and communicating results **99**
12.1 The ideas . . . 99
12.2 Example 1: Setting a proficiency standard . . . 100
12.3 Example 2: Telling the story . . . 102
12.4 Your turn . . . 104
In the lab . . . 105
Further reading . . . 105

I AI in measurement **106**

13 Machine learning through an IRT lens **107**
13.1 The ideas . . . 107
13.2 Example 1: The 2PL is a logistic regression on theta . . . 108
13.3 Example 2: Forty items or one score? . . . 110
13.4 Your turn . . . 111
In the AI Hub . . . 112
Further reading . . . 112

14 What item text tells us **113**
14.1 The ideas . . . 113
14.2 Example 1: Features and an explanatory model . . . 114
14.3 Example 2: The one-step LLTM in mirt . . . 117
14.4 Example 3: New items or new kinds of items? . . . 119
14.5 Your turn . . . 120
In the lab . . . 122
Further reading . . . 122

15 Generating items **123**
15.1 The ideas . . . 123
15.2 Example 1: Writing an item model . . . 124
15.3 Example 2: Calibrating the families . . . 126
15.4 Example 3: One step, with a random item effect . . . 128

15.5 Example 4: Does the family mean predict new items? . . . 129
15.6 Your turn . . . 130
In the lab . . . 132
Further reading . . . 132

16 Automated scoring **134**
16.1 The ideas . . . 134
16.2 Example 1: Training and judging an engine . . . 135
16.3 Example 2: What is the engine rewarding? . . . 138
16.4 Example 3: The engine on the Wright map . . . 139
16.5 Your turn . . . 143
In the lab . . . 145
Further reading . . . 145

17 Large language models as a colleague **146**
17.1 The ideas . . . 146
17.2 Example 1: From a sentence to a tested script . . . 147
17.3 Example 2: Writing items, and catching the mistakes . . . 148
17.4 Example 3: Is a recording reproducible? . . . 149
17.5 Where do synthetic respondents land? . . . 151
17.6 Your turn . . . 153
In the AI Hub . . . 154
Further reading . . . 154

18 Validity, fairness and governance of AI **155**
18.1 The ideas . . . 155
18.2 Example 1: Auditing a scoring engine . . . 156
18.3 Example 2: Other AI components . . . 159
18.4 Example 3: From audit to argument . . . 160
18.5 Governance . . . 162
18.6 Your turn . . . 162
In the lab . . . 164
Further reading . . . 164

II Knowledge graphs **165**

19 Knowledge graphs and IRT **166**
19.1 The ideas . . . 166
19.2 Example 1: A graph for fraction subtraction . . . 167
19.3 Example 2: From graph to IRT model . . . 169
19.4 Example 3: Checking the graph against data . . . 171
19.5 Example 4: Drafting a graph from item text . . . 175
19.6 Your turn . . . 177
In the lab . . . 179
Further reading . . . 180

20 Knowledge tracing **181**
20.1 The ideas . . . 181

20.2 Example 1: Plotting skills in a Cognitive Tutor . . . 182
20.3 Example 2: Knowledge tracing and IRT . . . 186
20.4 Example 3: Practice on a knowledge graph . . . 187
20.5 Your turn . . . 190
Further reading . . . 192

A Datasets used in the book **193**
Other datasets for practice . . . 197

B Adapting the book for your course **198**
The files . . . 198
Every chapter has the same shape . . . 199
Swapping in your own data . . . 199
Building the book . . . 199
Suggesting changes . . . 200

List of Figures

2.1 Item characteristic curves of the 2PL model for the five LSAT7 items. 26
2.2 Rasch curves (left) are parallel by construction; 2PL curves (right) can differ in slope. 27
2.3 Empirical item curve of the hardest driving-test item, D20, under the 2PL (left) and the 3PL (right). 30

3.1 Model comparison for LSAT6: each model's information criterion relative to the best model (0 = best). 34

4.1 Q3 residual correlations from a unidimensional 2PL fitted to the reading test with a four-item passage. 42

5.1 S-X^2 RMSEA of each SAT12 item under the Rasch model; flagged items are marked. 47
5.2 Empirical item curve of the worst-fitting item under the Rasch model (left) and the 2PL (right). 49
5.3 Item health card of the SAT12 item with the lowest discrimination under the 2PL. 50
5.4 Distribution of the Zh person-fit statistic for regular students, random responders and students with pre-knowledge. 51
5.5 Empirical item curve of the best-fitting item under the Rasch model. 53

6.1 Test information (left) and standard error of measurement (right) for the SAT12 science test under the 2PL. 55
6.2 Item information curves of six SAT12 items. 56
6.3 Test information of three 20-item forms drawn from the nursing pool; the dashed line marks the pass mark. 58

7.1 Distribution of the estimated scores (left) and the Wright map of people and items (right) for LSAT7. 63
7.2 True and estimated ability on the mathematics test; misclassified students lie near the cut. 65

8.1 Category response curves of the graded response model for the four Science items. 70
8.2 Expected item score curves of the graded response model for the Science items. 71
8.3 Wright map of the Science items: category thresholds next to the distribution of people. 71

8.4 Category response curves of J2 and J3; category 2 of J3 is never the most likely response. . . . 73

9.1 DIF effect sizes for the 25 reading items, paper versus tablet. . . . 79
9.2 Item curves for the paper and tablet groups: uniform DIF in R07, non-uniform DIF in R22. . . . 79

10.1 Anchor difficulties in the old and new calibrations; items off the line may have drifted. . . . 87
10.2 Test characteristic curves of the old and new forms on the common scale, with the cut score. . . . 90

11.1 Information of the 200-item vocabulary bank, stacked by content area. . . . 94
11.2 Number of items administered by the CAT against true ability. . . . 96

12.1 Item map of the ordered item booklet with the proficient cut score. . . . 102

13.1 Item slopes estimated by the 2PL and by logistic regression on the standardized rest score, against the true slopes. The dashed line marks perfect recovery. . . . 109

14.1 Cross-validated predicted against estimated difficulty for the three models of Example 1. Points on the dashed line are predicted exactly; labels mark the worst-predicted items. . . . 119

15.1 Item families after splitting the fraction model. Dots are items, diamonds family means, and bars the 95% interval for a new isomorph. . . . 128

16.1 Agreement report for the engine on the test set: QWK with the first rater (dot) next to human-human QWK (cross), and the machine-human SMD with the band of ± 0.10. . . . 138
16.2 Wright map with a rater facet: Students (left), the thresholds of the three written answers (middle), and the severity of the two human raters and the engine (right), all on one logit scale. The faint points are each rater's severity on each item. . . . 141
16.3 Wright map of human vs AI scoring: The students' scores (left) and the thresholds of each written answer (right) from human scores (circles) and machine scores (triangles). Thick lines mark thresholds that move by more than 0.3 logits. 143

17.1 Coverage strip for Honesty-Humility in Canada: Four language models, four attempts each. . . . 152

18.1 The score checks of the audit. Dashed lines are the default limits; bars are 95% confidence intervals. . . . 159

19.1 The fraction subtraction graph: Eight attributes on the left, each joined to the items that need it. . . . 168
19.2 The evidence on the three-concept graph: Partial correlations next to each pair, the suggested related edge as a dashed line, and the weak link of FS13 as a dotted line. . . . 173

20.1 Learning curves for the plotting skills: The mean probability of knowing each skill before each attempt (lines) and the proportion of correct answers (points). 184
20.2 One student's probability of knowing each plotting skill after every attempt: Filled points are correct answers. 185
20.3 The fractions graph shaded by the mean probability of knowing each concept after the last practice attempt. 189

List of Tables

1 The twelve chapters and their matching labs. 11
2 The chapters on AI in measurement. 12

A.1 Data chunks in the book and the objects each defines. 193

B.1 Source files of the book. 198

Preface

This book is a set of extended course notes on item response theory (IRT). It is written for instructors who teach measurement with R and for the students in their courses. Every chapter mixes a short explanation of the ideas with worked examples that you can run line by line, using the mirt package (Chalmers 2012) for estimation and mirtverse for model choice, tidy tables, graphics and plain-language interpretation.

How the book is organised

The twelve chapters follow the twelve weeks of the mirtverse course map, so an instructor can assign one chapter a week and pair it with the matching lab.

Table 1: The twelve chapters and their matching labs.

Chapter	Topic	Matching lab
1	Measurement, scoring and classical item analysis	Lab 1 (first two sections)
2	The Rasch, 2PL and 3PL models	
3	Estimation and comparing models	
4	Dimensionality and local independence	Lab 1
5	Item fit, person fit and item quality	
6	Information, precision and reliability	
7	Scoring people	Lab 2
8	Polytomous items and open-ended responses	Lab 6
9	Differential item functioning and fairness	Lab 3
10	Linking and equating	Lab 4

Chapter	Topic	Matching lab
11	Item banks, test assembly and adaptive testing	Lab 5
12	Cut scores, reporting and communicating results	Final project

A second part, *AI in measurement*, is for readers who know IRT and want to learn how machine learning and large language models fit into psychometrics. It goes with the mirtverse AI Hub and grows as the hub does.

Table 2: The chapters on AI in measurement.

Chapter	Topic	Matching lab
13	Machine learning through an IRT lens	
14	What item text tells us	Lab 8
15	Generating items	Lab 9
16	Automated scoring	Lab 7
17	Large language models as a colleague	
18	Validity, fairness and governance of AI	Lab 10

A third part, *Knowledge graphs*, describes what a test measures as a graph of concepts, prerequisites and items, and turns the graph into IRT models. Chapter 19 builds and checks the graphs, and Chapter 20 follows students' mastery of the concepts over time with knowledge tracing.

Every chapter has the same parts:

- **Learning goals:** What you should be able to do after the chapter.
- **Key terms:** Links to the mirtverse glossary.
- **The ideas:** A short explanation, with formulas only where they help.
- **Example 1 and Example 2:** Two worked analyses on different data, so you see the same method in two settings.
- **Your turn:** Exercises with solutions. In the online edition the solutions are folded away; click to open them.
- **In the lab:** The interactive mirtverse lab that goes with the chapter.
- **Further reading.**

The examples deliberately use different data from the labs. The labs hide problems in their data for students to find, and the book does not give those away.

Before you start

You need R (version 4.1 or later), mirt (version 1.48 or later) and mirtverse:

```
install.packages(c("mirt", "learnr"))
install.packages("mirtverse", repos = c("https://okanbulut.r-universe.dev",
                                        "https://cloud.r-project.org"))
library(mirtverse)
```

Loading mirtverse also loads mirt, so `mirt()`, `fscores()`, `itemfit()` and the mirt datasets are available straight away. Every example runs in a few seconds on a laptop, and no internet connection is needed once the packages are installed.

Students who are new to R can follow much of the book without writing code. The workbench (`launch_workbench()`) runs the same analyses by pointing and clicking and shows the R code for every step, and `mv_ask()` writes a script from a plain-language request.

Other teaching materials

- Teaching labs: Ten interactive learnr labs with an instructor guide and answer key.
- Course map: A 12-week syllabus with readings, functions, labs and assessment ideas.
- Glossary of IRT terms.
- Cheat sheet of the main functions, on two printable pages.
- Assessment pipeline: A test's full life cycle from blueprint to bank monitoring.

Using and adapting the book

The source of every chapter is an R Markdown file in the `book/` folder of the mirtverse repository. Each example loads its data in one clearly labelled chunk, so you can swap in your own data and rebuild the chapter. Appendix B explains how.

Other formats

The book is free to read online at https://www.okanbulut.com/mirtverse/book/. It is also available as a PDF for printing and an EPUB for e-readers; in the online edition, use the download button in the toolbar. All three are built from the same source each time the book changes.

How to cite this book

If you use the book in a course or in your work, please cite it as:

> Bulut, O. (2026). *Item response theory with mirt and mirtverse: Notes for teaching and learning*. Zenodo. https://doi.org/10.5281/zenodo.23885605

This is version 1.1 of the book. The DOI above always points to the latest edition. Each archived edition also has its own DOI, listed on the Zenodo record; cite that one if you need to refer to the exact text you read (version 1.0 is 10.5281/zenodo.23885606).

```
@book{bulut2026irt,
  author    = {Okan Bulut},
  title     = {Item Response Theory with mirt and mirtverse: Notes for Teaching and
   ↪  Learning},
  year      = {2026},
  publisher = {Zenodo},
  doi       = {10.5281/zenodo.23885605},
  url       = {https://www.okanbulut.com/mirtverse/book/}
}
```

Please also cite the software you use: mirt (Chalmers 2012) and mirtverse. Running `citation("mirtverse")` in R gives the mirtverse reference with its DOI.

Licence

© 2026 Okan Bulut. The text and figures of this book are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International licence (CC BY-NC-SA 4.0): You may share and adapt them for non-commercial purposes, such as teaching, as long as you give credit and share your adaptations under the same licence. The R code in the book is licensed under GPL-3, like mirtverse, so you can reuse it freely in your own analyses.

Acknowledgements

This book stands on the work of many people. Phil Chalmers wrote and maintains mirt, which does all of the estimation in these pages. The book was written with R Markdown and bookdown, by Yihui Xie and colleagues, and the figures were drawn with ggplot2. My thanks to all of them, and to the R community that makes open teaching materials like this possible.

About the author



Okan Bulut is a Professor of Measurement, Evaluation, and Data Science at the University of Alberta. His work sits where artificial intelligence meets educational measurement, data mining and learning analytics, and it asks a practical question: How can assessments learn about the learner while the learner is learning?

His research uses AI-driven algorithms and natural language processing to build digital assessment systems that adapt to learners' preferences, cognitive styles and performance over time. A large part of that work is about scale: Generating test items automatically, scoring open-ended responses automatically, and giving every student timely, useful feedback, not just the few whose work a teacher has time to mark by hand.

He works closely with international research teams, school districts and professional organizations to advance the responsible and ethical use of AI in education, and partners with industry and regulatory bodies to turn advances in psychometrics and learning analytics into practical assessment and feedback systems that support better educational decisions.

At the University of Alberta he teaches courses in educational measurement, assessment, computational psychometrics and machine learning, and he gives workshops on computational techniques, programming and data visualization. This book grew out of that teaching. mirtverse, the R package used throughout, is his attempt to make modern item response theory as approachable in the classroom as it is powerful in research.

- E-mail: bulut@ualberta.ca
- Website: http://www.okanbulut.com

- GitHub: https://github.com/okanbulut

Chapter 1

Measurement, scoring and classical item analysis

> **Learning goals.** After this chapter you can
> - turn raw responses into item scores with a key, and reverse-score negatively worded items;
> - compute and read proportion correct, item-total correlations and coefficient alpha;
> - use a distractor analysis to spot an item that is miskeyed or badly written;
> - say what a sum score can and cannot tell you, and why we need IRT.

Key terms: classical test theory, sum score, p-value, item-total correlation, dichotomous item, polytomous item.

1.1 The ideas

A test is a sample of tasks that we use to make an inference about something we cannot see directly: A student's reading ability, a nurse's clinical knowledge, a patient's level of anxiety. Before any model is fitted, raw responses have to be turned into **item scores**. For a multiple-choice item that means comparing the option chosen with the key (1 = right, 0 = wrong). For a rating scale it means coding the categories as 0, 1, 2, ... and making sure that a higher score always means "more" of the trait, which is why negatively worded items are reverse-scored.

The simplest summary of a person's performance is the **sum score**, the total of the item scores. **Classical test theory** (CTT) models the sum score as a true score plus error, $X = T + E$, and gives us a small toolkit for judging items (Crocker and Algina 1986):

- **Item difficulty** (p): The proportion of people who answer correctly (for rating items, the item mean). Confusingly, a *high* p means an *easy* item.
- **Item discrimination**: The correlation between the item score and the total score. Items that good performers get right and weak performers get wrong have high item-total

correlations. The *corrected* correlation removes the item from the total first, which matters for short tests.

- **Coefficient alpha**: An estimate of reliability, the proportion of sum-score variance that is true-score variance.
- **Distractor analysis**: For multiple-choice items, how often each wrong option is chosen, and by whom. A good distractor attracts low scorers. An option that attracts high scorers is a warning sign: It may be a second correct answer, or the key may be wrong.

CTT statistics are easy to compute and to explain, and you should always start with them. But they have two limits that motivate the rest of this book. First, p and the item-total correlation depend on the group that took the test: The same item looks easy in a strong group and hard in a weak one. Second, the sum score treats every item as equally informative and gives one standard error for everybody, even though a test measures some people more precisely than others. Item response theory addresses both.

1.2 Example 1: A multiple-choice science test

The `SAT12` data in mirt hold the answers of 600 high-school students to 32 multiple-choice science items with five options (8 marks an omitted item). The key is published on the help page, `?SAT12`.

```
# Raw multiple-choice responses (options 1-5, 8 = omitted) and the key.
raw <- SAT12
key <- c(1,4,5,2,3,1,2,1,3,1,2,4,2,1,5,3,4,4,1,4,3,3,4,1,3,5,1,3,1,5,4,5)
```

`key2binary()` scores every response against the key. Omits count as wrong.

```
resp <- key2binary(raw, key)
resp[1:5, 1:8]
#>      Item.1 Item.2 Item.3 Item.4 Item.5 Item.6 Item.7 Item.8
#> [1,]      1      1      1      1      1      1      1      1
#> [2,]      0      1      0      0      1      0      1      0
#> [3,]      1      1      1      0      1      0      1      0
#> [4,]      0      1      0      1      1      0      1      0
#> [5,]      0      1      1      1      1      0      1      1
total <- rowSums(resp)
summary(total)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>     4.0    14.8    18.0    18.2    22.0    32.0
```

`itemstats()` gives the classical statistics for every item in one table.

```
st <- itemstats(resp)
st$overall
#>    N mean.total sd.total std.alpha alpha SEM.alpha mean.r  sd.r
```

```
#>  600       18.2     5.05     0.795 0.798      2.27  0.108 0.075
head(st$itemstats, 8)
#>          N K  mean    sd cor_if_rm   smc alpha_if_rm
#> Item.1 600 2 0.283 0.451     0.300 0.158       0.793
#> Item.2 600 2 0.568 0.496     0.464 0.270       0.785
#> Item.3 600 2 0.280 0.449     0.371 0.184       0.789
#> Item.4 600 2 0.378 0.485     0.235 0.145       0.796
#> Item.5 600 2 0.620 0.486     0.340 0.197       0.791
#> Item.6 600 2 0.160 0.367     0.351 0.185       0.791
#> Item.7 600 2 0.760 0.427     0.289 0.200       0.793
#> Item.8 600 2 0.202 0.402     0.233 0.116       0.795
```

The `mean` column is the proportion correct, `cor_if_rm` is the corrected item-total correlation, and `alpha_if_rm` is alpha with the item removed. Let us sort the items by their corrected correlation and look at the weakest.

```
items <- st$itemstats
items[order(items$cor_if_rm), c("mean", "cor_if_rm", "alpha_if_rm")][1:5, ]
#>          mean cor_if_rm alpha_if_rm
#> Item.32 0.162     0.037       0.802
#> Item.12 0.415     0.076       0.803
#> Item.9  0.885     0.127       0.798
#> Item.21 0.915     0.136       0.798
#> Item.11 0.983     0.156       0.797
```

A corrected correlation near zero, or an alpha that goes *up* when the item is removed, means the item is not measuring what the rest of the test measures. Before blaming the item writer, check the key. A distractor analysis splits students into groups by total score and shows which option each group chose.

```
distractors <- function(raw, total, item, groups = 5) {
  band <- cut(rank(total, ties.method = "first"), groups,
              labels = c("lowest", paste0("band ", 2:(groups - 1)), "highest"))
  round(prop.table(table(band, option = raw[[item]]), 1), 2)
}
distractors(raw, total, "Item.32")
#>          option
#> band         1    2    3    4    5    8
#>   lowest  0.12 0.26 0.35 0.09 0.14 0.04
#>   band 2  0.10 0.31 0.36 0.07 0.14 0.02
#>   band 3  0.19 0.16 0.42 0.08 0.15 0.00
#>   band 4  0.16 0.12 0.48 0.08 0.16 0.00
#>   highest 0.06 0.07 0.62 0.04 0.22 0.00
```

The keyed answer to item 32 is option 5, yet option 3 is the most popular choice in every band, and its share climbs steeply from the lowest band to the highest. Option 5 gains far less. When an "unkeyed" option behaves the way a correct answer should, suspect the key. The help page for `SAT12` agrees: The key for item 32 is probably wrong and should be 3. Re-scoring fixes the item:

```
key_fixed <- replace(key, 32, 3)
resp_fixed <- key2binary(raw, key_fixed)
st_fixed <- itemstats(resp_fixed)
c(old_r = items["Item.32", "cor_if_rm"],
  new_r = st_fixed$itemstats["Item.32", "cor_if_rm"],
  old_alpha = st$overall$alpha, new_alpha = st_fixed$overall$alpha)
#>     old_r     new_r old_alpha new_alpha
#>    0.0371    0.2026    0.7979    0.8023
```

What to take away. A low item-total correlation is a symptom, not a diagnosis. The distractor table tells you *why*: A wrong key, two defensible answers, an option that confuses strong students, or simply an item that is too hard for everyone.

1.3 Example 2: A wellbeing questionnaire

Rating scales need the same care. Here we simulate a short questionnaire: 800 adults rate eight statements about their wellbeing from 0 (*strongly disagree*) to 3 (*strongly agree*). Items 3 and 7 are negatively worded ("I often feel tense"), so agreeing with them means *lower* wellbeing.

```
# Simulated 8-item wellbeing scale, 0-3, items W3 and W7 negatively worded.
set.seed(101)
a <- matrix(c(1.6, 1.4, 1.2, 1.8, 1.3, 1.5, 1.1, 0.5))
d <- matrix(c( 2.0, 0.5, -1.5,
               1.8, 0.2, -1.8,
               2.2, 0.8, -1.0,
               1.5, 0.0, -2.0,
               2.5, 1.0, -0.8,
               1.7, 0.3, -1.6,
               2.0, 0.6, -1.2,
               1.0, 0.0, -1.0), ncol = 3, byrow = TRUE)
resp <- simdata(a, d, N = 800, itemtype = "graded")
colnames(resp) <- paste0("W", 1:8)
resp[, c("W3", "W7")] <- 3 - resp[, c("W3", "W7")]   # as answered on the form
resp <- as.data.frame(resp)
```

If we forget to reverse-score, the two negative items correlate *negatively* with the total, and alpha is badly understated.

```
itemstats(resp)$itemstats[, c("mean", "cor_if_rm")]
#>    mean cor_if_rm
#> W1 1.64     0.348
#> W2 1.49     0.341
#> W3 1.24    -0.412
#> W4 1.44     0.326
#> W5 1.88     0.296
#> W6 1.60     0.356
#> W7 1.29    -0.433
#> W8 1.48     0.122
itemstats(resp)$overall$alpha
#> [1] 0.224
```

Reverse-scoring maps 0 to 3, 1 to 2 and so on:

```
scored <- resp
scored[, c("W3", "W7")] <- 3 - scored[, c("W3", "W7")]
st <- itemstats(scored)
st$itemstats[, c("mean", "cor_if_rm", "alpha_if_rm")]
#>    mean cor_if_rm alpha_if_rm
#> W1 1.64     0.543       0.710
#> W2 1.49     0.459       0.725
#> W3 1.76     0.451       0.727
#> W4 1.44     0.538       0.710
#> W5 1.88     0.443       0.728
#> W6 1.60     0.507       0.716
#> W7 1.71     0.473       0.723
#> W8 1.48     0.210       0.773
st$overall$alpha
#> [1] 0.753
```

All items now point the same way. Item W8 still has a low corrected correlation, and alpha rises from 0.75 to 0.77 without it: It is a weak item, and an IRT model will give it a low discrimination too (Chapter 2). For rating scales the category frequencies are worth a look as well. A category almost nobody uses carries little information and may need to be merged with its neighbour (Chapter 8).

```
round(st$proportions, 2)
#>       0    1    2    3
#> W1 0.19 0.23 0.34 0.24
#> W2 0.24 0.24 0.32 0.20
#> W3 0.16 0.22 0.33 0.30
#> W4 0.26 0.26 0.27 0.21
#> W5 0.12 0.21 0.35 0.32
```

```
#> W6 0.20 0.23 0.33 0.23
#> W7 0.17 0.22 0.34 0.27
#> W8 0.27 0.25 0.20 0.28
```

1.4 A first look with mirtverse

`profile_data()` gives a quick overview of any response file, and is what `mv_ask()` uses before it writes an analysis. For each column it reports the type, the number of distinct values, the range and the share of missing values, and it guesses which columns are items and which are IDs or grouping variables.

```
profile_data(scored)
#> Data profile: 800 rows, 8 columns (8 look like items)
#>  column    type n_distinct min max prop_missing role levels
#>      W1 integer          4   0   3            0 item
#>      W2 integer          4   0   3            0 item
#>      W3 integer          4   0   3            0 item
#>      W4 integer          4   0   3            0 item
#>      W5 integer          4   0   3            0 item
#>      W6 integer          4   0   3            0 item
#>      W7 integer          4   0   3            0 item
#>      W8 integer          4   0   3            0 item
```

1.5 Your turn

1. Score `SAT12` with the corrected key. Which three items are now the hardest, and which three have the lowest corrected item-total correlation?
2. Run `distractors()` on the item with the lowest corrected correlation. Is the problem the key, a distractor, or the difficulty of the item?
3. In Example 2, what happens to alpha if you drop W8? Would you drop it? Give one reason for and one against.
4. Suppose 30% of students omitted the last five items of `SAT12` because they ran out of time. Why would scoring omits as wrong distort the item statistics of those items?

Solutions

```
resp_fixed <- key2binary(SAT12, replace(key, 32, 3))
it <- itemstats(resp_fixed)$itemstats
rownames(it)[order(it$mean)][1:3]                # hardest
#> [1] "Item.6" "Item.8" "Item.3"
rownames(it)[order(it$cor_if_rm)][1:3]        # weakest
#> [1] "Item.12" "Item.9"  "Item.21"
```

2. Use `distractors(SAT12, rowSums(resp_fixed), "<item>")`. If no option is preferred by the top band and the correct option rises only slowly, the item is simply hard or vague rather than miskeyed.
3. Alpha rises only a little. Dropping W8 shortens the scale and may remove content the scale is meant to cover; keeping it costs little precision. A content review should decide, not alpha alone.
4. Not-reached items would look harder and would correlate with speed rather than knowledge. Treat not-reached items as missing (`NA`) rather than wrong; IRT estimation handles missing responses naturally.

In the lab

> The first two sections of Lab 1, `run_lab("model-choice")`, put you in charge of scoring a certification exam with 1,200 candidates. You score the responses, screen the items and decide what to do about anything suspicious. See Teaching labs.

Further reading

Crocker and Algina (1986) remain the classic introduction to classical test theory and item analysis. The *Standards for Educational and Psychological Testing* (American Educational Research Association et al. 2014) set out what evidence a test score needs before it is used.

Chapter 2

The Rasch, 2PL and 3PL models

Learning goals. After this chapter you can

- explain what a latent trait is and why IRT puts people and items on the same scale;
- write down the Rasch, 2PL and 3PL models and say what each parameter does to the item characteristic curve;
- fit the three models with `mirt()` and read their parameters in the traditional IRT metric;
- describe an item in words from its curve.

Key terms: item response theory, latent trait, logit, Rasch model, 2PL, 3PL, item characteristic curve, difficulty, discrimination, guessing, slope-intercept form.

2.1 The ideas

IRT assumes that a person's responses depend on an unobserved **latent trait** θ (theta): Ability, knowledge, an attitude. By convention θ has mean 0 and standard deviation 1 in the calibration group, so a person at $\theta = 1$ is one standard deviation above average.

Each item gets an **item characteristic curve** (ICC): The probability of a correct answer as a function of θ. The three classic models differ in how many parameters shape that curve (Hambleton et al. 1991; Embretson and Reise 2000).

The Rasch (1PL) model has one parameter per item, the difficulty b_j:

$$P(X_{ij} = 1 \mid \theta_i) = \frac{\exp(\theta_i - b_j)}{1 + \exp(\theta_i - b_j)}.$$

The difficulty is the point on the θ scale where the probability is 0.5. All curves have the same slope, so they never cross, and the sum score contains all the information about θ.

The two-parameter logistic (2PL) model adds a discrimination a_j:

$$P(X_{ij} = 1 \mid \theta_i) = \frac{\exp[a_j(\theta_i - b_j)]}{1 + \exp[a_j(\theta_i - b_j)]}.$$

A larger a_j gives a steeper curve: The item separates people just below b_j from people just above it more sharply. Values between about 0.8 and 2.5 are typical for good achievement items; values below about 0.5 mean the item hardly tells people apart.

The three-parameter logistic (3PL) model adds a lower asymptote c_j, often called the guessing parameter:

$$P(X_{ij} = 1 \mid \theta_i) = c_j + (1 - c_j)\frac{\exp[a_j(\theta_i - b_j)]}{1 + \exp[a_j(\theta_i - b_j)]}.$$

Even very low-ability people get a multiple-choice item right with probability c_j, roughly one over the number of options if they guess at random. With a lower asymptote, b_j is where the probability is $(1 + c_j)/2$ rather than 0.5.

mirt estimates the models in **slope-intercept form**, $a_j\theta_i + d_j$, which behaves better numerically. The two forms are related by $b_j = -d_j/a_j$. Use `coef(mod, IRTpars = TRUE)`, or mirtverse's `tidy()` (which uses `IRTpars = TRUE` by default), to see a and b.

2.2 Example 1: The LSAT section 7 items

`LSAT7` holds the response patterns of 1,000 people to five items from a section of the Law School Admission Test, stored as a frequency table. `expand.table()` turns it into one row per person.

```
# Five dichotomous LSAT items, one row per test taker.
resp <- expand.table(LSAT7)
```

We fit the three models. `itemtype` chooses the model and `verbose = FALSE` hides the iteration history.

```
rasch <- mirt(resp, 1, itemtype = "Rasch", verbose = FALSE)
twopl <- mirt(resp, 1, itemtype = "2PL",   verbose = FALSE)
threepl <- mirt(resp, 1, itemtype = "3PL", verbose = FALSE)
```

The item parameters in the IRT metric:

```
coef(twopl, IRTpars = TRUE, simplify = TRUE)$items
#>            a      b g u
#> Item.1 0.988 -1.879 0 1
#> Item.2 1.081 -0.748 0 1
#> Item.3 1.706 -1.058 0 1
#> Item.4 0.765 -0.635 0 1
#> Item.5 0.736 -2.520 0 1
```

The columns `g` and `u` are the lower and upper asymptotes, fixed at 0 and 1 for the 2PL. The same information as a tidy table, ready for a report:

```
tidy(twopl)
#> # A tibble: 10 x 5
#>    item   itemtype term  estimate std.error
#>    <chr>  <chr>    <chr>    <dbl>     <dbl>
#>  1 Item.1 2PL      a        0.988        NA
#>  2 Item.1 2PL      b       -1.88         NA
#>  3 Item.2 2PL      a        1.08         NA
#>  4 Item.2 2PL      b       -0.748        NA
#>  5 Item.3 2PL      a        1.71         NA
#>  6 Item.3 2PL      b       -1.06         NA
#>  7 Item.4 2PL      a        0.765        NA
#>  8 Item.4 2PL      b       -0.635        NA
#>  9 Item.5 2PL      a        0.736        NA
#> 10 Item.5 2PL      b       -2.52         NA
```

All five items have negative difficulties: This is an easy section for this group. Item 3 has the steepest slope, items 4 and 5 the flattest. The curves show it at a glance.

```
gg_trace(twopl, facet = FALSE)
```

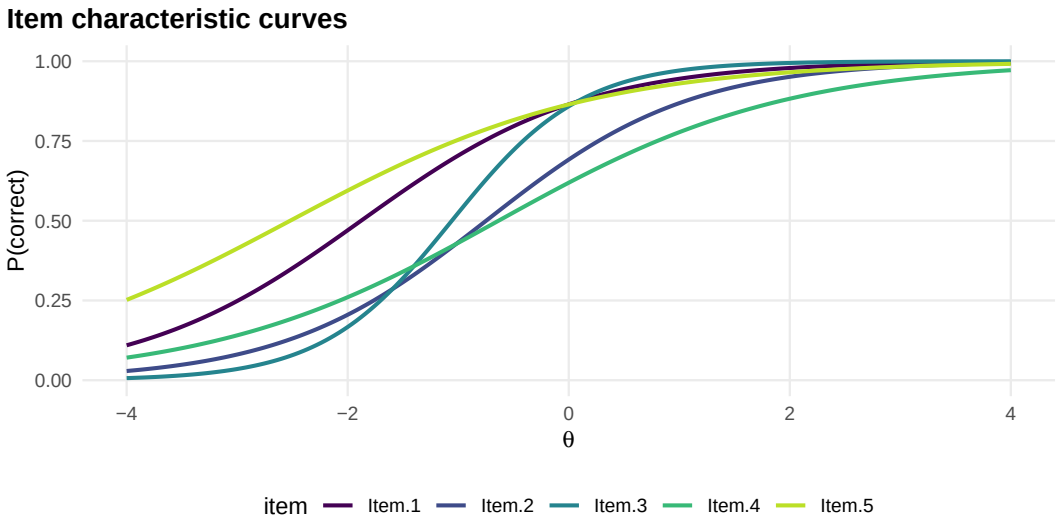


Figure 2.1: Item characteristic curves of the 2PL model for the five LSAT7 items.

Compare the Rasch curves, which are parallel by construction, with the 2PL curves, which are not:

```
library(patchwork)
(gg_trace(rasch, facet = FALSE) + ggplot2::ggtitle("Rasch")) +
```

```
(gg_trace(twopl, facet = FALSE) + ggplot2::ggtitle("2PL")) +
plot_layout(guides = "collect")
```

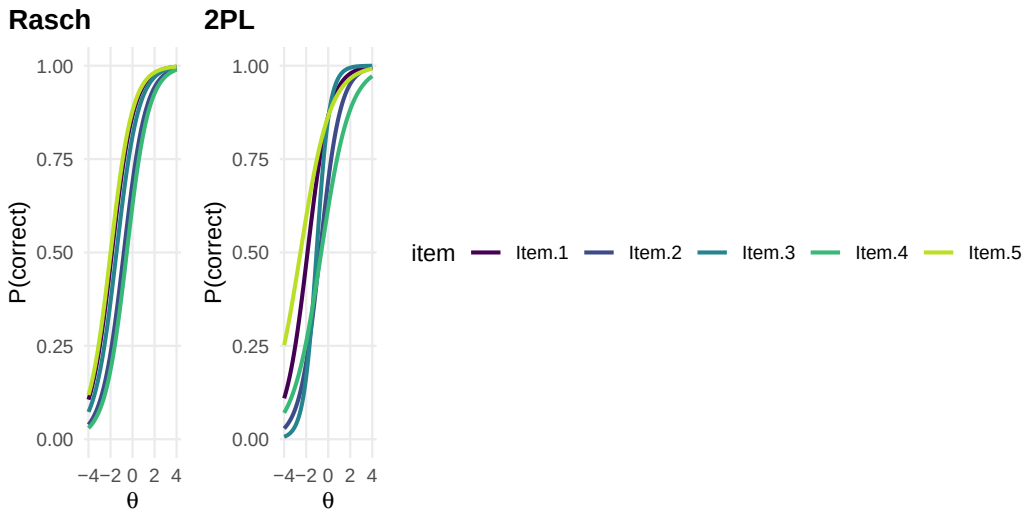


Figure 2.2: Rasch curves (left) are parallel by construction; 2PL curves (right) can differ in slope.

The 3PL adds a lower asymptote for each item. With only five easy items there is very little information about what low-ability people do. Look at the c estimates (`g`) and their confidence intervals:

```
threepl_se <- mirt(resp, 1, itemtype = "3PL", SE = TRUE, verbose = FALSE)
tidy(threepl_se, conf.int = TRUE) |> subset(term == "g")
#> # A tibble: 5 x 7
#>   item   itemtype term   estimate std.error conf.low conf.high
#>   <chr>  <chr>    <chr>     <dbl>     <dbl>    <dbl>     <dbl>
#> 1 Item.1 3PL      g     0.0000703   0.00930 -0.0182    0.0183
#> 2 Item.2 3PL      g     0.295       0.140    0.0209    0.569
#> 3 Item.3 3PL      g     0.000258    0.0278  -0.0543    0.0548
#> 4 Item.4 3PL      g     0.0000506   0.00435 -0.00847   0.00857
#> 5 Item.5 3PL      g     0.000155    0.0209  -0.0408    0.0411
```

Four of the five lower asymptotes sit at essentially zero, on the boundary of the parameter space, and the one that does not (Item 2) has a confidence interval that runs from almost 0 to well over 0.5. The data say very little about guessing. Which model should we report? That is a question for Chapter 3. For now, notice that a model with more parameters is not automatically better: It has to be supported by the data.

`interpret()` describes the fitted model in plain language. Its *Items* section is a useful check on your own reading of the curves.

```
interpret(twopl)
#> Overview
#> --------
#> * A 2PL model with 1 dimension was fitted to 1000 respondents and 5 items (EM
#>   estimation).
#> * The estimation converged normally.
#>
#> Model fit
#> ---------
#> * Overall model fit is close: M2 = 11.94, df = 5, RMSEA = 0.037, CFI = 0.968,
#>   SRMSR = 0.032.
#> * The chi-square test is significant, but with large samples this is common;
#>   the RMSEA indicates the misfit is small in practical terms.
#>
#> Reliability
#> -----------
#> * Marginal reliability is 0.44 (low).
#> * Scores are too imprecise for decisions about individuals; consider adding
#>   items or using the scores only for group-level summaries.
#>
#> Targeting
#> ---------
#> * The average item location is -1.37 and the average person score is 0.00, so
#>   the test is relatively easy for this group.
#> * Measurement is most precise around theta = -1.08 (information = 1.47, SE =
#>   0.83).
#>
#> Items
#> -----
#> * Low item-total correlation: Item.5 (r = 0.17). These items relate weakly to
#>   the rest of the test.
#> * Misfit: Item.2 (S-X2 p < .001). The model does not describe responses to
#>   these items well (alpha = 0.01); inspect them with gg_empirical() or
#>   item_card().
#>
#> Local dependence
#> ----------------
#> * No item pairs show signs of local dependence (Q3).
#>
#> Person fit
#> ----------
#> * 2.1% of respondents have unusual response patterns (Zh < -1.96), which is
#>   within what chance alone would produce.
```

2.3 Example 2: A driving theory test with guessing

To see when the 3PL matters, we simulate a 20-item driving theory test with four options per question, taken by 3,000 learner drivers. Because we simulate, we know the true parameters and can check how well each model recovers them.

```
# Simulated 20-item, four-option test with real guessing (c around 0.2).
set.seed(202)
n_items <- 20
true <- data.frame(a = rlnorm(n_items, 0.2, 0.25),
                   b = seq(-1.5, 2, length.out = n_items),
                   c = rbeta(n_items, 5, 17))
resp <- simdata(a = matrix(true$a), d = matrix(-true$a * true$b),
                N = 3000, itemtype = "3PL", guess = true$c)
colnames(resp) <- sprintf("D%02d", 1:n_items)
```

The data chunk converts the true b into an intercept with $d = -ab$, because `simdata()` uses the slope-intercept form.

```
fit2 <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
fit3 <- mirt(resp, 1, itemtype = "3PL", verbose = FALSE)
est2 <- coef(fit2, IRTpars = TRUE, simplify = TRUE)$items
est3 <- coef(fit3, IRTpars = TRUE, simplify = TRUE)$items
recovery <- data.frame(
  model = rep(c("2PL", "3PL"), each = 2),
  parameter = rep(c("a", "b"), 2),
  bias = c(mean(est2[, "a"] - true$a), mean(est2[, "b"] - true$b),
           mean(est3[, "a"] - true$a), mean(est3[, "b"] - true$b)),
  rmse = c(sqrt(mean((est2[, "a"] - true$a)^2)), sqrt(mean((est2[, "b"] -
   ↪  true$b)^2)),
           sqrt(mean((est3[, "a"] - true$a)^2)), sqrt(mean((est3[, "b"] -
            ↪  true$b)^2))))
recovery
#>   model parameter    bias  rmse
#> 1   2PL         a -0.4681 0.583
#> 2   2PL         b -0.5731 0.698
#> 3   3PL         a  0.0622 0.338
#> 4   3PL         b -0.0188 0.379
```

When people really do guess, the 2PL absorbs the guessing into the other parameters: It **underestimates discrimination** and makes hard items look **easier** than they are. The 3PL recovers the parameters much better, and its lower asymptotes are close to the truth on average:

```
c(mean_true_c = mean(true$c), mean_estimated_c = mean(est3[, "g"]))
#>      mean_true_c mean_estimated_c
```

```
#>           0.239           0.226
```

The empirical item curve makes the problem visible. For the hardest item, the observed proportions correct for low-ability groups sit well above the 2PL curve, which heads towards zero:

```
gg_empirical(fit2, "D20") + gg_empirical(fit3, "D20")
```

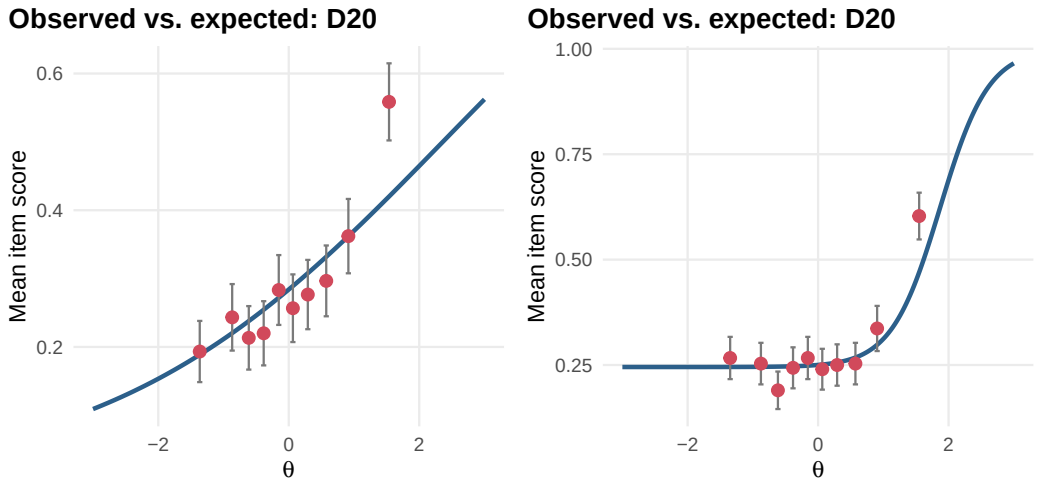


Figure 2.3: Empirical item curve of the hardest driving-test item, D20, under the 2PL (left) and the 3PL (right).

What to take away. The 3PL is worth its extra parameters when items can be answered by guessing *and* there are enough low-ability test takers to estimate the lower asymptote: Here 3,000 people and a range of difficulties. With small samples or easy tests, c is poorly determined, and a 2PL (or a 3PL with a prior on c) is often the safer choice.

2.4 Your turn

1. Using the 2PL for `LSAT7`, compute by hand the probability that a person at $\theta = -1$ answers Item 3 correctly. Check your answer with `probtrace(extract.item(twopl, 3), -1)`.
2. Fit the Rasch model to `LSAT7` and compare its difficulties with the 2PL difficulties. Which item moves most, and why?
3. In Example 2, refit the 3PL using only the first 300 test takers. What happens to the recovery of c?
4. Sketch (on paper) the curve of an item with $a = 0.3$ and $b = 0$. Would you keep this item on a test used to award licences?

Solutions

```
p <- coef(twopl, IRTpars = TRUE, simplify = TRUE)$items
plogis(p["Item.3", "a"] * (-1 - p["Item.3", "b"]))
#> [1] 0.525
probtrace(extract.item(twopl, 3), matrix(-1))
#>        P.0   P.1
#> [1,] 0.475 0.525

pr <- coef(rasch, IRTpars = TRUE, simplify = TRUE)$items
round(cbind(rasch = pr[, "b"], twopl = p[, "b"]), 2)
#>        rasch twopl
#> Item.1 -1.87 -1.88
#> Item.2 -0.79 -0.75
#> Item.3 -1.46 -1.06
#> Item.4 -0.52 -0.64
#> Item.5 -1.99 -2.52

small <- mirt(resp[1:300, ], 1, itemtype = "3PL", verbose = FALSE)
summary(coef(small, IRTpars = TRUE, simplify = TRUE)$items[, "g"] - true$c)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> -0.3425 -0.0895  0.0558  0.0270  0.1615  0.3461
```

2. Items whose 2PL slope is far from the common Rasch slope move most, here Item 3 (the steepest) and Item 5 (one of the flattest). A flat item (small a) has a 2PL difficulty further from zero, because its curve only reaches 0.5 far out on the scale.
3. You may first see a warning that the EM algorithm stopped after 500 cycles: The 3PL is hard to estimate in small samples (Chapter 3). With 300 people the c estimates scatter much more widely around the truth, and some may be close to 0 or implausibly large. The estimates of a and b get worse too, because the three parameters trade off.
4. The curve is almost flat: The chance of a correct answer barely depends on ability. Such an item adds little information and some noise; it should go back to the item writers.

In the lab

> Lab 1, `run_lab("model-choice")`, asks you to compare the Rasch, 2PL and 3PL models for a real-sized certification exam and to defend your choice to the board. Chapter 3 gives you the tools for that comparison.

Further reading

Embretson and Reise (2000) give an accessible introduction for psychologists; Hambleton, Swaminathan and Rogers (1991) a short classic; de Ayala (2022) a thorough modern treatment. Chalmers (2012) describes the mirt package.

Chapter 3

Estimation and comparing models

Learning goals. After this chapter you can

- describe marginal maximum likelihood estimation in words;
- check whether a model has converged, and know what to do when it has not;
- compare nested models with a likelihood-ratio test and any models with information criteria;
- judge overall model fit with M2, RMSEA, SRMSR and CFI;
- explain why AIC and BIC can disagree, and what sample size has to do with it.

Key terms: calibration, marginal maximum likelihood, convergence, standard error, likelihood-ratio test, information criteria, M2, RMSEA, SRMSR, CFI and TLI.

3.1 The ideas

Calibration means estimating the item parameters. We never observe θ, so we cannot simply regress responses on it. **Marginal maximum likelihood** (MML) instead assumes a distribution for θ in the population, usually standard normal, and integrates it out: The likelihood of a response pattern is its probability averaged over all possible values of θ. The item parameters are chosen to make the observed patterns as likely as possible (Bock and Aitkin 1981).

mirt maximizes this likelihood with the **EM algorithm** by default. It replaces the integral with a sum over a grid of *quadrature points* and alternates two steps: The E-step works out, for every person, how likely each grid point is given their responses and the current item parameters; the M-step updates each item's parameters as if those expected counts were data. The cycles stop when the parameters stop changing, which is called **convergence**. If the algorithm hits its limit of 500 cycles first, mirt warns you, and the estimates should not be trusted.

Once a model has converged, we compare it with alternatives.

- **Likelihood-ratio test (LRT).** For *nested* models (the Rasch model is a 2PL with equal slopes; the 2PL is a 3PL with $c = 0$), twice the difference in log-likelihoods is approx-

imately chi-square distributed with degrees of freedom equal to the difference in the number of parameters. With large samples even trivial improvements become significant.

- **Information criteria.** AIC $= -2\log L + 2k$ and BIC $= -2\log L + k\log N$, where k is the number of parameters and N the sample size. Smaller is better. BIC punishes extra parameters more heavily once $N > 7$, so it favours simpler models. A difference of 10 or more BIC points is usually read as strong evidence.
- **Absolute fit.** The **M2** statistic (Maydeu-Olivares and Joe 2006) compares observed and expected proportions for single items and pairs of items. RMSEA (below about 0.05 is close fit), SRMSR (below about 0.05) and CFI (above about 0.95) are effect sizes built from it. Treat the cut-offs as rough guides, not rules.

3.2 Example 1: A test the Rasch model fits

`LSAT6` is another five-item section of the Law School Admission Test, with 1,000 test takers. It is a textbook example of data the Rasch model fits well.

```
# Five dichotomous LSAT items, one row per test taker.
resp <- expand.table(LSAT6)
```

Fit the models one at a time, check convergence, and compare them:

```
rasch <- mirt(resp, 1, itemtype = "Rasch", verbose = FALSE)
twopl <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
extract.mirt(rasch, "converged")
#> [1] TRUE
extract.mirt(twopl, "iterations")
#> [1] 22
anova(rasch, twopl)
#>        AIC SABIC   HQ  BIC logLik    X2 df     p
#> rasch 4946  4956 4957 4975  -2467
#> twopl 4953  4971 4972 5002  -2467 0.569  4 0.966
```

The 2PL improves the log-likelihood by a fraction of a point for four extra parameters, so the LRT is far from significant and both AIC and BIC prefer the Rasch model. `glance()` adds the absolute fit statistics:

```
glance(rasch)[, c("logLik", "AIC", "BIC", "M2", "df", "p.value", "RMSEA", "SRMSR",
 ↪  "CFI")]
#> # A tibble: 1 x 9
#>   logLik   AIC   BIC    M2    df p.value RMSEA  SRMSR   CFI
#>    <dbl> <dbl> <dbl> <dbl> <int>   <dbl> <dbl>  <dbl> <dbl>
#> 1 -2467. 4946. 4975.  5.29     9   0.808     0 0.0224     1
```

`select_model()` runs all of this in one call. It fits the Rasch, 2PL and 3PL models (or the

polytomous equivalents), checks convergence, ranks them by BIC, runs the LRTs between nested pairs and explains its recommendation.

```
sel <- select_model(resp, verbose = FALSE)
sel
#> mirtverse model selection (ranked by BIC)
#>
#>  model npar logLik  AIC  BIC SABIC RMSEA CFI delta weight
#>  Rasch    6  -2467 4946 4975  4956     0   1   0.0      1
#>    2PL   10  -2467 4953 5002  4971     0   1  27.1      0
#>    3PL   15  -2467 4963 5037  4989    NA  NA  61.6      0
#>
#> The Rasch model is recommended because it has the lowest BIC (4975.3). The
#> runner-up, 2PL, is 27.1 BIC points worse, which is very strong evidence for the
#> Rasch model. Likelihood-ratio test Rasch vs. 2PL: chi-square(4) = 0.57, p =
#> .966; the extra parameters of 2PL are not needed. Likelihood-ratio test Rasch
#> vs. 3PL: chi-square(9) = 0.54, p = 1.000; the extra parameters of 3PL are not
#> needed. Absolute fit of the Rasch model: RMSEA = 0.000 (close fit), CFI =
#> 1.000, SRMSR = 0.022.
```

```
autoplot(sel)
```

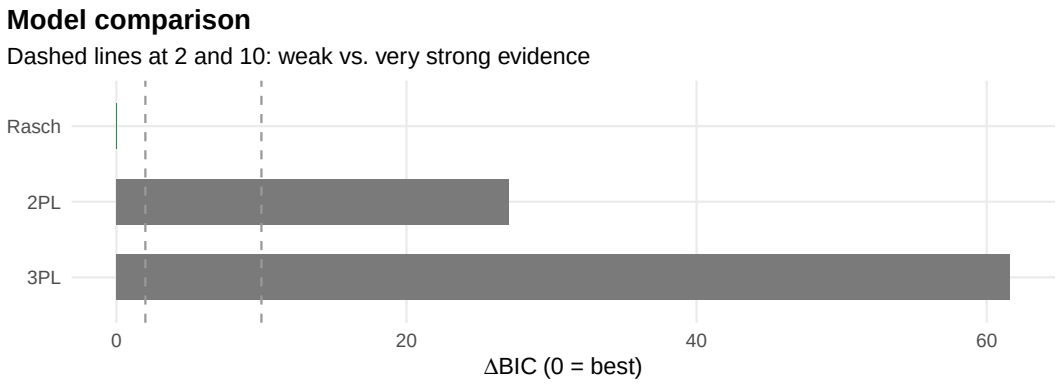


Figure 3.1: Model comparison for LSAT6: each model's information criterion relative to the best model (0 = best).

The `weight` column converts BIC differences into the relative support for each model; here nearly all of it goes to the Rasch model.

3.3 Example 2: When the criteria disagree

Now take the 32 science items of `SAT12`, scored with the corrected key from Chapter 1.

```
# SAT12 science items, scored with the corrected key (item 32 keyed 3).
key <- c(1,4,5,2,3,1,2,1,3,1,2,4,2,1,5,3,4,4,1,4,3,3,4,1,3,5,1,3,1,5,4,3)
resp <- key2binary(SAT12, key)
```

```
sel <- select_model(resp, verbose = FALSE)
sel
#> mirtverse model selection (ranked by BIC)
#>
#>  model npar logLik   AIC   BIC SABIC RMSEA   CFI delta weight
#>    2PL   64  -9622 19373 19654 19451 0.029 0.953   0.0      1
#>  Rasch   33  -9768 19602 19747 19642 0.039 0.905  92.5      0
#>    3PL   96  -9572 19337 19759 19454 0.022 0.975 104.5      0
#>
#> The 2PL model is recommended because it has the lowest BIC (19654.2). The
#> runner-up, Rasch, is 92.5 BIC points worse, which is very strong evidence for
#> the 2PL model. Likelihood-ratio test Rasch vs. 2PL: chi-square(31) = 290.83, p
#> < .001; the 2PL model fits significantly better. Likelihood-ratio test 2PL vs.
#> 3PL: chi-square(32) = 100.16, p < .001; the 3PL model fits significantly
#> better. Note the tension: the likelihood-ratio test prefers 3PL, while BIC
#> penalizes extra parameters and prefers 2PL. Choose 2PL if parsimony matters
#> most, or 3PL if fit to these data matters most. AIC would instead select the
#> 3PL model. Absolute fit of the 2PL model: RMSEA = 0.029 (close fit), CFI =
#> 0.953, SRMSR = 0.045.
sel$lrt
#> # A tibble: 3 x 5
#>   simpler complex    X2    df  p.value
#>   <chr>   <chr>   <dbl> <int>    <dbl>
#> 1 Rasch   2PL      291.    31 5.30e-44
#> 2 Rasch   3PL      391.    63 7.62e-49
#> 3 2PL     3PL      100.    32 6.00e- 9
```

This time the evidence is mixed. The LRT and AIC prefer the 3PL, while BIC prefers the 2PL. Neither is "wrong": They answer different questions. The LRT asks whether the extra 32 guessing parameters improve fit beyond chance, and with 600 people the answer is yes. BIC asks which model is more likely to be the better description once we account for the cost of 32 more parameters, and it says the 2PL. A sensible report states the tension and the reason for the final choice: If scores will be used for decisions about low-scoring students, where guessing matters most, the 3PL may be worth its cost; for a short research scale, the 2PL is easier to defend.

3.3.1 Small samples and convergence

The 3PL is the hardest of the three to estimate, because the lower asymptote is determined only by what low-ability people do. With 200 students the EM algorithm may not converge at

all:

```
set.seed(303)
small <- resp[sample(nrow(resp), 200), ]
fit_small <- mirt(small, 1, itemtype = "3PL", verbose = FALSE)
extract.mirt(fit_small, "converged")
#> [1] FALSE
summary(coef(fit_small, IRTpars = TRUE, simplify = TRUE)$items[, "g"])
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> 0.000276 0.002136 0.079776 0.125231 0.207500 0.500034
```

Some lower asymptotes drift to zero and others to implausibly large values. Three common remedies are to use a 2PL, to fix c at a sensible value (for five-option items, 0.2), or to put a **prior** on c so that the data only move it away from a plausible value when they have something to say. In mirt a prior is part of the model syntax. Here a beta(5, 17) prior centres c near 0.2 (`expbeta` applies it on the scale mirt uses internally):

```
spec <- mirt.model("F = 1-32
                    PRIOR = (1-32, g, expbeta, 5, 17)")
fit_prior <- mirt(small, spec, itemtype = "3PL", verbose = FALSE)
#> Warning: Observed log-likelihood may still be increasing.
#>   Final log-likelihood values were:
#>   -3154.689, -3151.159, -3149.755
extract.mirt(fit_prior, "converged")
#> [1] TRUE
summary(coef(fit_prior, IRTpars = TRUE, simplify = TRUE)$items[, "g"])
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   0.109   0.181   0.192   0.187   0.195   0.235
```

With the prior the model converges and the lower asymptotes stay in a believable range. Report priors when you use them: They are part of the model.

3.4 Your turn

1. For `LSAT6`, compute the LRT statistic for Rasch versus 2PL by hand from the two log-likelihoods, and check it against `anova()`.
2. Fit the 2PL to `SAT12` with `quadpts = 15` and with the default of 61 points. Do the item parameters change? What does that tell you?
3. Draw 300 students from `SAT12` and rerun `select_model()`. Does the recommendation change? Repeat with another random sample.
4. A colleague reports "the 3PL fitted significantly better ($p < .001$), so we used it". What would you ask them?

Solutions

```r
ll <- c(extract.mirt(rasch, "logLik"), extract.mirt(twopl, "logLik"))
lrt <- 2 * (ll[2] - ll[1])
c(X2 = lrt, df = 4, p = pchisq(lrt, 4, lower.tail = FALSE))
#>    X2    df     p
#> 0.569 4.000 0.966

m15 <- mirt(resp, 1, "2PL", quadpts = 15, verbose = FALSE)
m61 <- mirt(resp, 1, "2PL", quadpts = 61, verbose = FALSE)
max(abs(coef(m15, simplify = TRUE)$items - coef(m61, simplify = TRUE)$items))
#> [1] 0.142
```

2. With only 15 points some estimates move by about 0.1 on the slope-intercept scale; between 61 (the default for one dimension) and more points they hardly move at all. Fewer points trade accuracy for speed, which matters mainly for models with several dimensions.
3. With fewer people, BIC's penalty matters relatively more and the likelihood gains of the 3PL are smaller, so the 2PL (or even the Rasch model) is chosen more often. Model choice depends on the sample, not only on the items.
4. Did the model converge? What did BIC say? How large is the sample? Are the guessing estimates plausible, and what were their standard errors? Would the scores or decisions differ between the 2PL and the 3PL?

In the lab

> Lab 1, `run_lab("model-choice")`, ends with a model choice you have to justify to a certification board, using the same evidence as this chapter: Convergence, likelihood-ratio tests, information criteria and absolute fit.

Further reading

Bock and Aitkin (1981) introduced the EM approach to marginal maximum likelihood. Maydeu-Olivares and Joe (2006) describe M2. de Ayala (2022) discusses model comparison and fit in detail.

Chapter 4

Dimensionality and local independence

Learning goals. After this chapter you can

- explain the assumptions of unidimensionality and local independence;
- compare one- and two-factor models and read rotated factor loadings;
- fit a confirmatory model in which each item measures a named dimension;
- read standardized residuals and Yen's Q3, and tell local dependence from a second trait;
- choose a remedy for locally dependent items.

Key terms: dimensionality, local independence, local dependence, Q3, multidimensional IRT.

4.1 The ideas

The models in Chapter 2 make two related assumptions. **Unidimensionality** says that one latent trait accounts for the relationships among the items. **Local independence** says that once we know a person's θ, their responses to different items are independent: The only reason two items correlate is that both depend on θ.

Both can fail, in different ways.

- A test may measure **two or more traits**, for example reading and arithmetic in a set of mathematics word problems. Then a one-factor model forces two traits into one score, and the items of the smaller dimension look poorly discriminating.
- A **small cluster of items** may share something beyond the trait: A common reading passage, a shared stimulus, near-identical wording, or an item whose answer gives away the next. These pairs or clusters are *locally dependent*. They look like extra-good items, they inflate reliability, and they distort the item parameters.

The tools differ too. For the first problem we compare models with one and two factors and look at the factor loadings. For the second we look at what is left over after fitting a one-factor model: If local independence holds, the residual correlations between item pairs should be

close to zero. Yen's **Q3** statistic (Yen 1984) is the correlation between two items' residuals. Values well above the average Q3 (often by 0.2 or more) flag dependent pairs.

4.2 Example 1: A mathematics test with a reading dimension

We simulate a 24-item mathematics test taken by 1,500 Grade 8 students. Items M01 to M12 are calculation items. Items W01 to W12 are word problems that also depend on reading, which correlates 0.5 with calculation skill.

```
# Simulated 24-item test, two correlated traits (r = 0.5), 12 items each.
set.seed(404)
a <- cbind(c(runif(12, 1.0, 2.0), runif(12, 0.3, 0.6)),
           c(rep(0, 12),          runif(12, 1.0, 1.8)))
d <- matrix(rnorm(24, 0, 0.8))
sigma <- matrix(c(1, 0.5, 0.5, 1), 2)
resp <- simdata(a, d, N = 1500, itemtype = "2PL", sigma = sigma)
colnames(resp) <- c(sprintf("M%02d", 1:12), sprintf("W%02d", 1:12))
```

First, an exploratory comparison of one and two factors. We keep a copy of the data as `math` for the exercises.

```
math <- resp
f1 <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
f2 <- mirt(resp, 2, itemtype = "2PL", verbose = FALSE)
anova(f1, f2)
#>      AIC SABIC    HQ   BIC logLik      X2 df p
#> f1 41008 41111 41103 41263 -20456
#> f2 40185 40337 40326 40563 -20022 868.777 23 0
```

The two-factor model is far better on every criterion. `summary()` shows the standardized loadings after an oblique rotation, which lets the factors correlate:

```
summary(f2, rotate = "oblimin", suppress = 0.25)
#>
#> Rotation:  oblimin
#>
#> Rotated factor loadings:
#>
#>        F1    F2    h2
#> M01       0.708 0.509
#> M02       0.717 0.504
#> M03       0.663 0.453
#> M04       0.628 0.453
#> M05       0.581 0.360
#> M06       0.598 0.394
```

```
#> M07       0.665 0.398
#> M08       0.559 0.281
#> M09       0.473 0.233
#> M10       0.617 0.390
#> M11       0.715 0.539
#> M12       0.724 0.492
#> W01 0.634       0.428
#> W02 0.520       0.430
#> W03 0.751       0.572
#> W04 0.646       0.444
#> W05 0.588       0.373
#> W06 0.737       0.512
#> W07 0.560       0.451
#> W08 0.785       0.600
#> W09 0.697       0.484
#> W10 0.766       0.570
#> W11 0.775       0.555
#> W12 0.716       0.486
#>
#> Rotated SS loadings:  5.67 5
#>
#> Factor correlations:
#>
#>       F1 F2
#> F1 1.000
#> F2 0.654  1
```

The calculation items load on one factor and the word problems on the other. The factor correlation is higher than the 0.5 we simulated between the two traits, because the word problems also need calculation skill: The rotation cannot separate that shared part cleanly, so some of it shows up as correlation between the factors. The one-factor fit statistics give the same message from the other side:

```
rbind(one_factor = glance(f1)[, c("RMSEA", "SRMSR", "CFI")],
      two_factor = glance(f2)[, c("RMSEA", "SRMSR", "CFI")])
#> # A tibble: 2 x 3
#>    RMSEA  SRMSR   CFI
#> *  <dbl>  <dbl> <dbl>
#> 1 0.0644 0.0518 0.936
#> 2 0      0.0173 1
```

Exploratory models are useful for finding structure. When the structure is known from the test blueprint, a **confirmatory** model states it directly. Each item loads only on its own factor, and the factor correlation is estimated (`COV = F1*F2`):

```
spec <- mirt.model("Calc    = 1-12
                    Reading = 13-24
                    COV     = Calc*Reading")
cfa <- mirt(resp, spec, itemtype = "2PL", verbose = FALSE)
coef(cfa, simplify = TRUE)$cov
#>          Calc Reading
#> Calc    1.000   0.676
#> Reading 0.676   1.000
anova(cfa, f2)
#>       AIC SABIC    HQ   BIC logLik     X2 df     p
#> cfa 40174 40279 40271 40434 -20038
#> f2  40185 40337 40326 40563 -20022 32.658 22 0.067
```

The confirmatory model is nearly as good as the exploratory one with far fewer parameters, so BIC prefers it. What should we report to the school? That depends on use. Two subscores are defensible if each is reliable enough (Chapter 6); a single total score mixes two skills, and students with weak reading will look weaker at mathematics than they are.

4.3 Example 2: A reading passage that ties four items together

In this test of 20 items, items R05 to R08 all refer to the same passage. Students who happen to understand the passage well do better on all four than their general reading ability predicts. We simulate this with a small extra factor that only those four items share.

```
# Simulated 20-item reading test; R05-R08 share a passage (testlet effect).
set.seed(405)
a_gen <- runif(20, 1.0, 1.8)
a_pas <- c(rep(0, 4), rep(2.0, 4), rep(0, 12))
resp <- simdata(cbind(a_gen, a_pas), matrix(rnorm(20, 0, 0.7)),
                N = 1200, itemtype = "2PL")
colnames(resp) <- sprintf("R%02d", 1:20)
```

We fit an ordinary unidimensional 2PL, as an analyst who did not know about the passage would, and look at the Q3 matrix.

```
uni <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
gg_q3(uni)
```

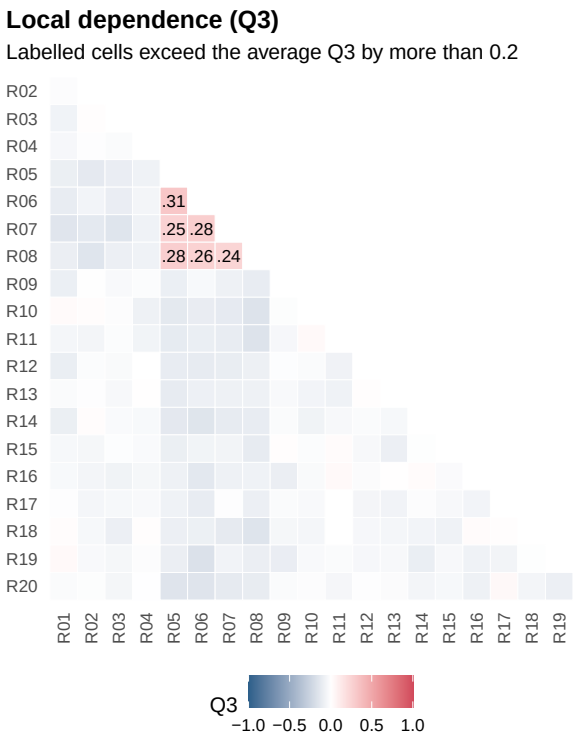


Figure 4.1: Q3 residual correlations from a unidimensional 2PL fitted to the reading test with a four-item passage.

The six pairs among R05 to R08 stand out. Most other Q3 values are slightly negative, which is normal: Residuals are a little negatively correlated on average, so we judge a pair against the average Q3 rather than against zero. The numbers behind the plot:

```
q3 <- residuals(uni, type = "Q3", verbose = FALSE)
round(q3[5:8, 5:8], 2)
#>      R05  R06  R07  R08
#> R05 1.00 0.31 0.25 0.28
#> R06 0.31 1.00 0.28 0.26
#> R07 0.25 0.28 1.00 0.24
#> R08 0.28 0.26 0.24 1.00
```

Local dependence has a cost. The unidimensional model has nowhere to put the passage effect, so it bends the item parameters to accommodate it and treats part of the passage effect as reading ability. Remedy 1 below shows how different the passage items look once the dependence is modelled.

There are three common remedies.

1. **Model the dependence.** A *testlet* or *bifactor* model gives the passage items a specific factor of their own, uncorrelated with the general factor. `bfactor()` fits it; its second argument says which specific factor each item belongs to (`NA` for none).

```
specific <- c(rep(NA, 4), rep(1, 4), rep(NA, 12))
bf <- bfactor(resp, specific, verbose = FALSE)
anova(uni, bf)
#>       AIC SABIC    HQ   BIC logLik      X2 df p
#> uni 28458 28534 28534 28661 -14189
#> bf  27901 27985 27985 28125 -13906 564.869  4 0
round(coef(bf, simplify = TRUE)$items[3:10, c("a1", "a2")], 2)
#>       a1   a2
#> R03 1.51 0.00
#> R04 0.93 0.00
#> R05 1.44 2.11
#> R06 1.45 2.20
#> R07 1.27 1.76
#> R08 1.88 1.99
#> R09 1.38 0.00
#> R10 1.31 0.00
```

The passage effect now has its own slope (`a2`), and the general slopes (`a1`) of R05 to R08 are higher than in the unidimensional model, which had diluted them. Compare them with `coef(uni, simplify = TRUE)$items[5:8, "a1"]`. Likelihood-ratio test and BIC both favour the bifactor model.

2. **Combine the items.** Sum R05 to R08 into one polytomous "super-item" scored 0 to 4 and fit a polytomous model (Chapter 8). This is simple and robust, at the cost of some information.
3. **Drop or rewrite items** in a future form, so that no two items depend on each other beyond the trait.

4.4 Your turn

1. In Example 1, compute the sum score of the word problems and of the calculation items separately. How strongly do they correlate? Why is this lower than the factor correlation?
2. Use `select_model(resp, dims = 1:2)` on the data of Example 1. Which model does it choose?
3. In Example 2, compare the empirical reliability of the scores (`empirical_rxx()` on `fscores(..., full.scores.SE = TRUE)`) from the unidimensional model with that of the general factor of the bifactor model. Which one would you report, and why?
4. Make the super-item from R05 to R08, fit a graded model to the 17 items, and check the Q3 matrix again.

Solutions

```
cor(rowSums(math[, 1:12]), rowSums(math[, 13:24]))
#> [1] 0.554
```

1. Sum scores contain measurement error, which attenuates (weakens) the correlation; the factor correlation is corrected for it.

```
select_model(math, dims = 1:2, verbose = FALSE)
#> mirtverse model selection (ranked by BIC)
#>
#>     model npar logLik   AIC   BIC SABIC RMSEA   CFI delta weight
#>  2PL (2F)   71 -20022 40185 40563 40337 0.000 1.000     0      1
#>     Rasch   25 -20526 41102 41235 41156 0.064 0.931   673      0
#>       2PL   48 -20456 41008 41263 41111 0.064 0.936   701      0
#>       3PL   72 -20450 41044 41426 41198 0.064 0.942   864      0
#>
#> The 2PL (2F) model is recommended because it has the lowest BIC (40562.7). The
#> runner-up, Rasch, is 672.6 BIC points worse, which is very strong evidence for
#> the 2PL (2F) model. Likelihood-ratio test Rasch vs. 2PL (2F): chi-square(46) =
#> 1008.97, p < .001; the 2PL (2F) model fits significantly better.
#> Likelihood-ratio test 2PL vs. 2PL (2F): chi-square(23) = 868.78, p < .001; the
#> 2PL (2F) model fits significantly better. Absolute fit of the 2PL (2F) model:
#> RMSEA = 0.000 (close fit), CFI = 1.000, SRMSR = 0.017.
```

2. It fits one- and two-factor versions of each candidate and should prefer a two-factor model, because the second dimension is strong.

```
empirical_rxx(fscores(uni, full.scores.SE = TRUE))
#>   F1
#> 0.85
empirical_rxx(fscores(bf, full.scores.SE = TRUE))
#>     G    S1
#> 0.838 0.587
```

3. The unidimensional model reports a somewhat higher reliability, because it counts the shared passage effect as signal about reading. The general factor of the bifactor model gives the more honest figure. With one small passage the difference is modest; with several passages it grows.

```
super <- data.frame(resp[, -(5:8)], R05_08 = rowSums(resp[, 5:8]))
sup <- mirt(super, 1, itemtype = c(rep("2PL", 16), "graded"), verbose = FALSE)
q3s <- residuals(sup, type = "Q3", verbose = FALSE)
max(abs(q3s[upper.tri(q3s)]))
#> [1] 0.131
```

4. The super-item removes the dependent pairs, and the largest remaining Q3 is small.

In the lab

> Lab 1, `run_lab("model-choice")`, includes a dimensionality check and a local dependence screen on a certification exam. Decide for yourself whether anything needs fixing before you choose a model.

Further reading

Yen (1984) introduced Q3 and showed how local dependence affects parameter estimates and equating. Embretson and Reise (2000) cover multidimensional and bifactor models.

Chapter 5

Item fit, person fit and item quality

Learning goals. After this chapter you can

- explain what item fit means and why it is always fit *to a model*;
- compute and read the S-X^2 item-fit statistic and its RMSEA;
- compare an empirical item curve with the model curve;
- flag aberrant response patterns with the Zh person-fit statistic, and say what can cause them;
- summarize an item's health for a review meeting.

Key terms: item fit, S-X^2, empirical item curve, person fit, Zh, model fit.

5.1 The ideas

Chapter 3 asked whether a model fits the test *as a whole*. Item fit asks the same question one item at a time: Does the item's curve describe how people at different ability levels actually responded?

The most widely used statistic is Orlando and Thissen's **S-X^2** (Orlando and Thissen 2000). It groups people by their sum score, compares the observed proportion correct in each group with the proportion the model predicts, and adds up the squared differences. A small p-value means the item does not follow the model. Because a test has many items, some will be flagged by chance, so it is good practice to adjust the p-values (for example with the Benjamini-Hochberg method) and to look at an effect size too: mirt reports an RMSEA for each item, where values below about 0.05 mean the misfit is small in practice.

Item fit is always fit *to a particular model*. An item with an unusually steep curve fits the 2PL perfectly well but misfits the Rasch model, which forces all slopes to be equal. A misfitting item is a question, not a verdict: Look at the empirical curve, the content and the key before you decide.

Person fit turns the question around: Does a person's response pattern look like what the

model expects of someone at their θ? The **Zh** statistic (Drasgow et al. 1985) is a standardized log-likelihood of the pattern. Values near zero are typical; large negative values (below about -2) mean the pattern is unlikely: A strong student who misses easy items, or a weak one who gets the hardest items right. Causes include careless responding, guessing, cheating, misaligned answer sheets, fatigue and atypical instruction. Zh flags a pattern for a closer look; it never proves misconduct.

5.2 Example 1: The same items under two models

We return to the `SAT12` science items, scored with the corrected key.

```
# SAT12 science items, scored with the corrected key (item 32 keyed 3).
key <- c(1,4,5,2,3,1,2,1,3,1,2,4,2,1,5,3,4,4,1,4,3,3,4,1,3,5,1,3,1,5,4,3)
resp <- key2binary(SAT12, key)
```

First, the Rasch model. `gg_itemfit()` plots each item's S-X² RMSEA and marks the items whose p-value is below `alpha`.

```
rasch <- mirt(resp, 1, itemtype = "Rasch", verbose = FALSE)
gg_itemfit(rasch, p_adjust = "BH")
```

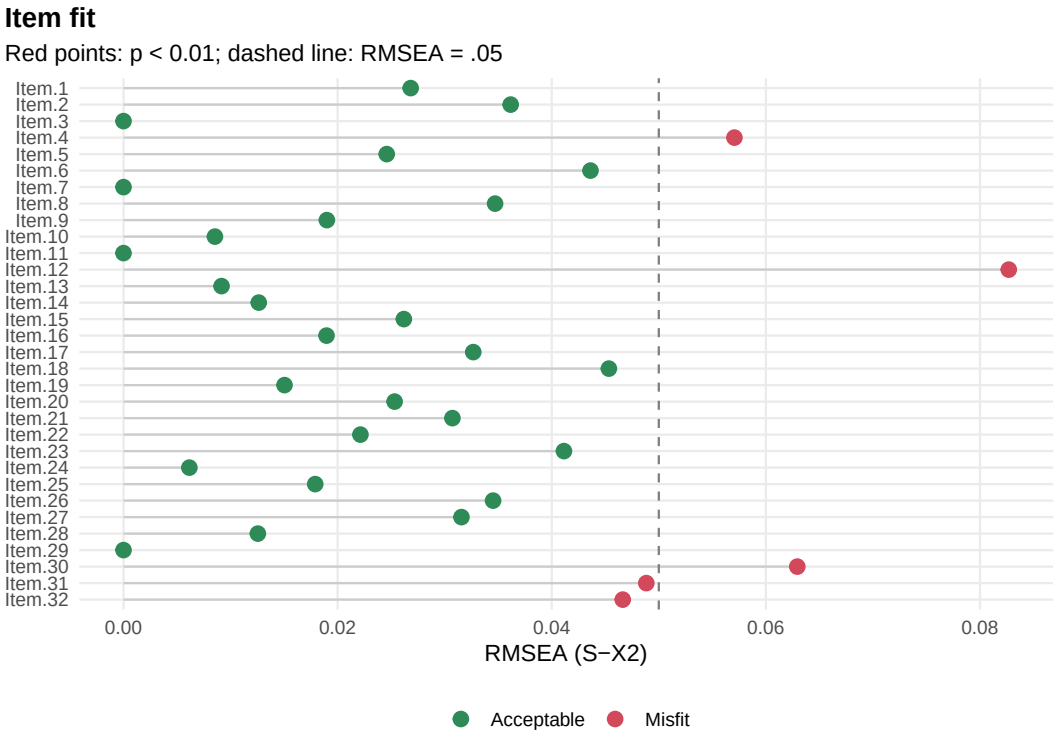


Figure 5.1: S-X^2 RMSEA of each SAT12 item under the Rasch model; flagged items are marked.

`item_table()` puts the classical statistics, IRT parameters, fit and flags in one table. Here are the items that misfit the Rasch model most:

```
tab_r <- item_table(rasch, p_adjust = "BH")
tab_r[order(tab_r$p.value), c("item", "item_total_r", "location", "S_X2", "df",
                              "RMSEA", "p.value")][1:5, ]
#> # A tibble: 5 x 7
#>   item    item_total_r location  S_X2    df  RMSEA  p.value
#>   <chr>          <dbl>    <dbl> <dbl> <dbl>  <dbl>    <dbl>
#> 1 Item.12       0.0685    0.394  91.7    18 0.0827 2.26e-10
#> 2 Item.30       0.157     0.274  60.7    18 0.0629 2.52e- 5
#> 3 Item.4        0.236     0.574  53.1    18 0.0571 2.69e- 4
#> 4 Item.31       0.425    -1.85   41.3    17 0.0488 6.80e- 3
#> 5 Item.32       0.203     0.258  41.4    18 0.0466 8.45e- 3
```

The worst-fitting items include items whose item-total correlation is far from the typical value: Either much flatter or much steeper than the common slope the Rasch model imposes. Now fit the 2PL, which lets the slopes vary:

```
twopl <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
tab_2 <- item_table(twopl, p_adjust = "BH")
c(misfit_rasch = sum(tab_r$p.value < 0.01), misfit_2pl = sum(tab_2$p.value < 0.01))
#> misfit_rasch   misfit_2pl
#>            5            0
```

The 2PL removes the misfit. Whether that is the right response depends on the purpose. A Rasch programme would rather revise or drop the items that do not fit, because equal slopes give the sum score its simple meaning; a 2PL programme would keep them.

An **empirical item curve** shows the misfit directly. People are grouped by their estimated θ, and the observed proportion correct in each group is plotted against the model curve:

```
worst <- tab_r$item[which.min(tab_r$p.value)]
gg_empirical(rasch, worst) + gg_empirical(twopl, worst)
```

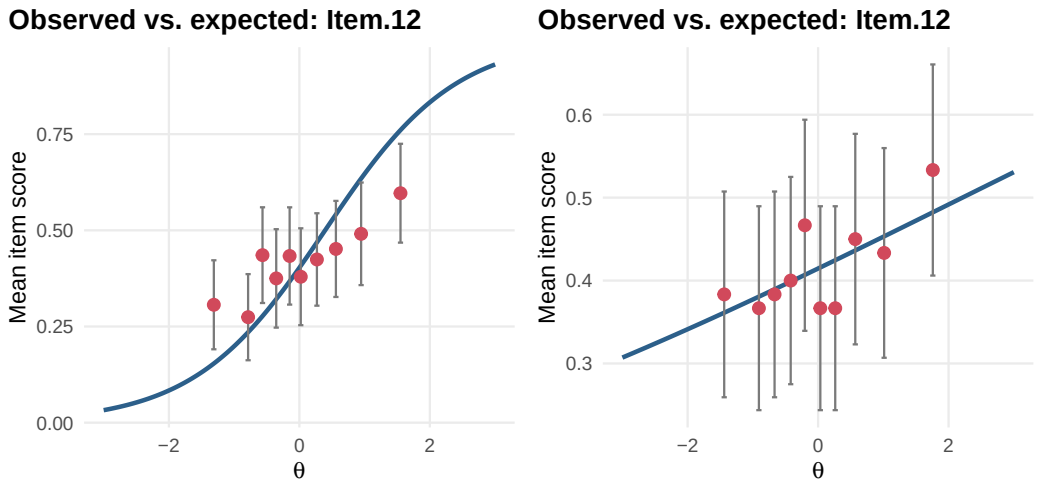


Figure 5.2: Empirical item curve of the worst-fitting item under the Rasch model (left) and the 2PL (right).

Under the Rasch model the observed points do not follow the curve; under the 2PL they do. Fit is not the only thing that matters about an item, though. `item_table()` also flags items that are very easy or very hard, or that discriminate poorly:

```
tab_2[tab_2$flag != "", c("item", "mean", "item_total_r", "discrimination",
                          "location", "flag")]
#> # A tibble: 7 x 6
#>   item     mean item_total_r discrimination location flag
#>   <chr>   <dbl>        <dbl>          <dbl>    <dbl> <chr>
#> 1 Item.9  0.885       0.123           0.518   -4.13  extreme location; lo~
#> 2 Item.11 0.983       0.150           1.69    -3.07  extreme location; lo~
#> 3 Item.12 0.415       0.0685          0.156    2.21  low discrimination; ~
#> 4 Item.17 0.963       0.199           1.53    -2.71  low item-total corre~
#> 5 Item.21 0.915       0.131           0.600   -4.19  extreme location; lo~
#> 6 Item.30 0.44        0.157           0.383    0.647 low discrimination; ~
#> 7 Item.32 0.443       0.203           0.490    0.484 low discrimination
```

An **item health card** collects everything about one item on a page, with a traffic-light status, for item review meetings:

```
weak <- tab_2$item[which.min(tab_2$discrimination)]
item_card(twopl, weak)
```

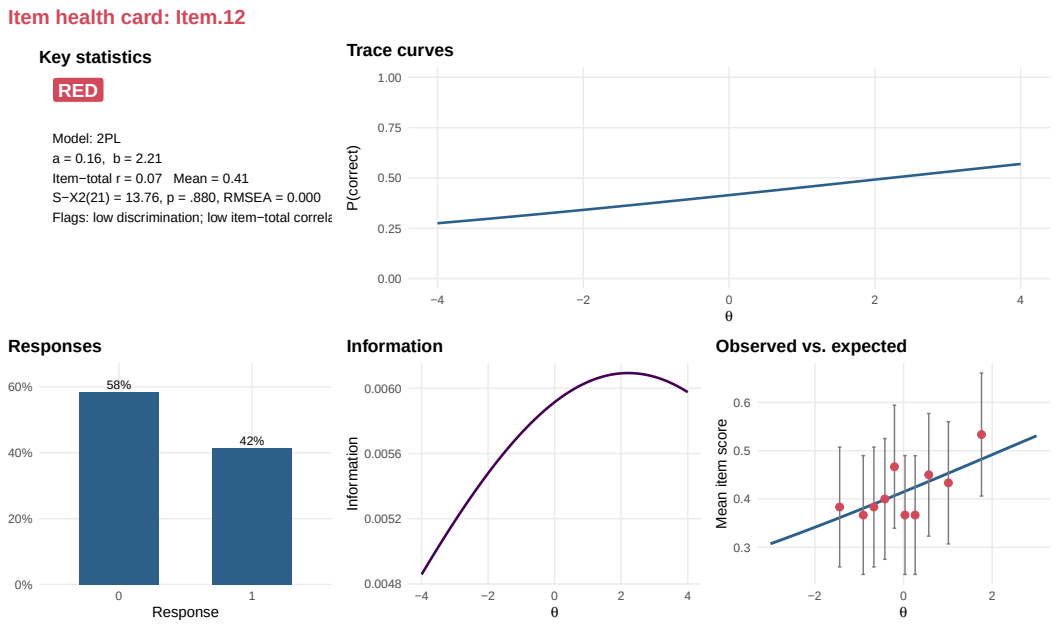


Figure 5.3: Item health card of the SAT12 item with the lowest discrimination under the 2PL.

`item_cards(twopl, file = "cards.html")` writes a card for every item into one HTML page that reviewers can open in a browser.

5.3 Example 2: Finding aberrant response patterns

We simulate a 30-item quiz taken by 1,000 students. Most students respond according to a 2PL model, but 40 answered at random (careless responding) and 30 weak students somehow got the ten hardest items right (perhaps they saw them in advance). We record who is who so we can check the person-fit statistic.

```
# Simulated 30-item quiz: 930 regular students, 40 random responders and
# 30 weak students who know the 10 hardest items. `type` records which.
set.seed(505)
a <- matrix(rlnorm(30, 0.2, 0.2)); b <- seq(-2, 2, length.out = 30)
resp <- simdata(a, matrix(-a * b), N = 1000, itemtype = "2PL")
colnames(resp) <- sprintf("Q%02d", 1:30)
type <- rep(c("regular", "random", "preknowledge"), c(930, 40, 30))
resp[type == "random", ] <- rbinom(40 * 30, 1, 0.5)
weak <- simdata(a, matrix(-a * b), N = 30, itemtype = "2PL",
                Theta = matrix(rnorm(30, -1.2, 0.3)))
weak[, 21:30] <- 1
resp[type == "preknowledge", ] <- weak
```

`augment()` adds each person's score, its standard error and, with `person_fit = TRUE`, the Zh statistic to the data:

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
scores <- augment(mod, person_fit = TRUE)
scores$type <- type
head(scores[, c(".F1", ".se_F1", ".Zh", "type")])
#> # A tibble: 6 x 4
#>       .F1 .se_F1    .Zh type
#>     <dbl>  <dbl>  <dbl> <chr>
#> 1 -1.74    0.472  0.166 regular
#> 2  1.48    0.406  0.970 regular
#> 3 -0.422   0.388  0.809 regular
#> 4 -0.0768  0.376 -0.238 regular
#> 5  1.01    0.380  1.03  regular
#> 6 -0.822   0.407 -0.218 regular
```

How well does Zh separate the groups?

```
library(ggplot2)
ggplot(scores, aes(.Zh, fill = type)) +
  geom_histogram(bins = 40, position = "identity", alpha = 0.6) +
  geom_vline(xintercept = -2, linetype = 2) +
  labs(x = "Zh person-fit statistic", y = "Students", fill = NULL) +
  theme_mirtverse()
```

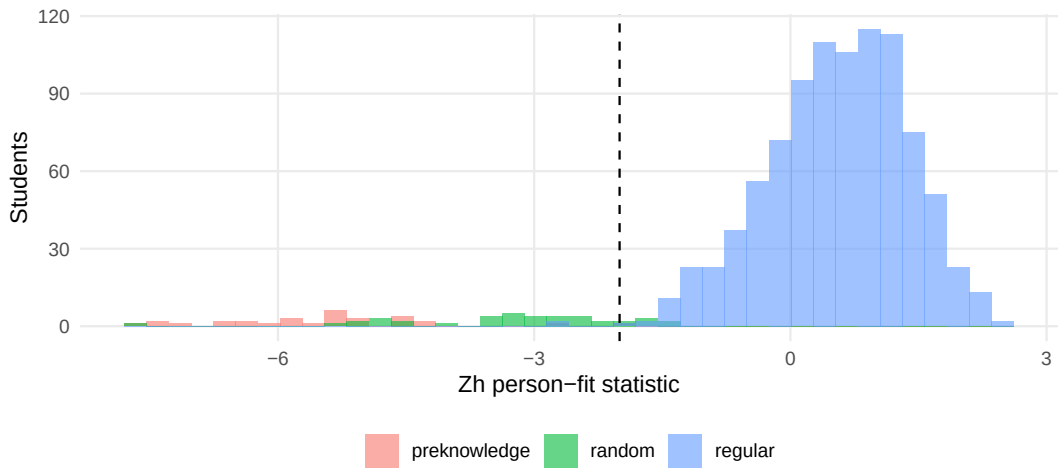


Figure 5.4: Distribution of the Zh person-fit statistic for regular students, random responders and students with pre-knowledge.

```
aggregate(.Zh < -2 ~ type, data = scores, FUN = mean)
#>           type .Zh < -2
#> 1 preknowledge  1.00000
#> 2       random  0.85000
```

```
#> 3      regular  0.00323
```

All of the students with pre-knowledge are flagged: Getting the hardest items right while missing easier ones is very unlikely under the model. Most random responders are flagged too, because on a 30-item test with a wide range of difficulties a coin-flip pattern contradicts the model again and again. Only a handful of regular students fall below -2. Real data are rarely this clean: Aberrance is usually milder, and only part of a test may be affected (the last items, say, for a student who ran out of time). That is why Zh is a screening tool: Flagged patterns need a human look, and often other evidence such as response times (see the response-time article).

Do the aberrant patterns distort the item parameters? We can check by refitting without the flagged students and comparing both sets of difficulties for the ten hardest items with the values used to simulate the data:

```
clean <- mirt(resp[scores$.Zh >= -2, ], 1, itemtype = "2PL", verbose = FALSE)
b_all <- coef(mod, IRTpars = TRUE, simplify = TRUE)$items[21:30, "b"]
b_clean <- coef(clean, IRTpars = TRUE, simplify = TRUE)$items[21:30, "b"]
round(c(error_all = mean(abs(b_all - b[21:30])),
        error_clean = mean(abs(b_clean - b[21:30]))), 3)
#>   error_all error_clean
#>       0.145       0.140
```

Here the difference is small, because only 7% of the students are aberrant and the model is estimated from everyone else. With more contamination, or with pre-knowledge concentrated on a few items, the damage grows, and removing flagged cases before calibration is common practice.

5.4 Your turn

1. Plot the empirical curve of the best-fitting item under the Rasch model in Example 1. How does it compare with the worst-fitting one?
2. Rerun `gg_itemfit()` for the Rasch model without the p-value adjustment. How many more items are flagged? Which result would you report?
3. In Example 2, which value of the Zh cut-off flags at most 5% of the regular students? How many students with pre-knowledge does it catch?
4. Write two sentences to a school principal explaining what a flagged person-fit statistic means and what it does not mean.

Solutions

```
best <- tab_r$item[which.max(tab_r$p.value)]
gg_empirical(rasch, best)
```

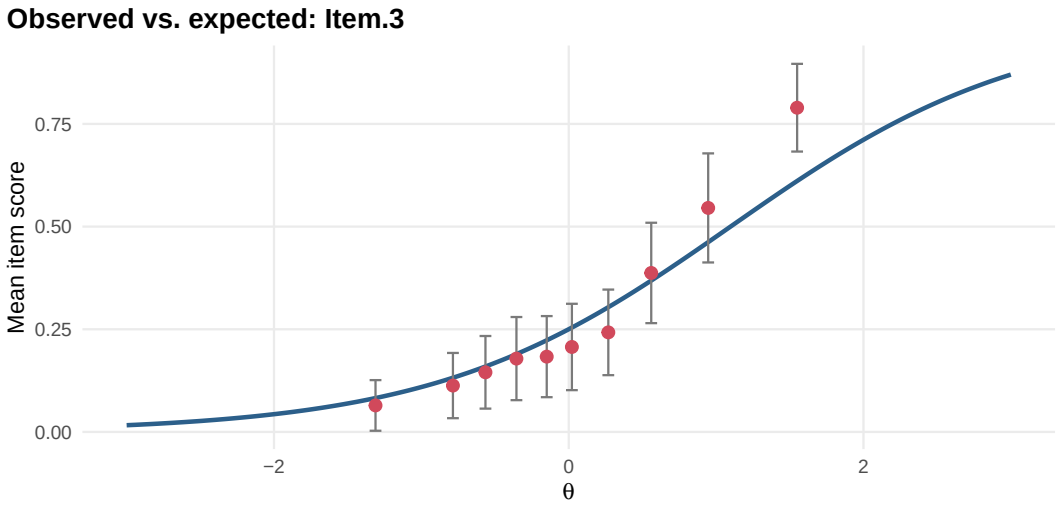


Figure 5.5: Empirical item curve of the best-fitting item under the Rasch model.

```
sum(item_table(rasch)$p.value < 0.01)
#> [1] 8
cut5 <- quantile(scores$.Zh[type == "regular"], 0.05)
c(cutoff = unname(cut5),
  caught = mean(scores$.Zh[type == "preknowledge"] < cut5))
#> cutoff caught
#> -0.945  1.000
```

2. Without adjustment more items are flagged, some only by chance. With 32 tests, report the adjusted results, and look at the RMSEA to judge size.
3. For example: "This student's pattern of right and wrong answers is unusual for someone with their overall score, so we checked it more closely. It is not evidence of cheating on its own; it can also come from guessing, fatigue or a mismatch between what was taught and what was tested."

In the lab

> Lab 1, `run_lab("model-choice")`, includes an item fit step in which you decide which items, if any, a certification board should drop or send back to the item writers.

Further reading

Orlando and Thissen (2000) introduced S-X². Drasgow, Levine and Williams (1985) introduced the standardized person-fit index Zh. de Ayala (2022) discusses item and person fit at length.

Chapter 6

Information, precision and reliability

Learning goals. After this chapter you can

- read item and test information curves;
- convert information into a conditional standard error of measurement;
- compute marginal and empirical reliability, and explain why a single reliability coefficient hides how precision changes along the scale;
- use information to judge whether a test is well targeted for a decision.

Key terms: item information, test information, standard error of measurement, marginal reliability, targeting, precision.

6.1 The ideas

In classical test theory a test has one standard error of measurement for everyone. In IRT, precision depends on where you are on the scale. An item tells us most about people whose ability is close to its difficulty: For them the outcome is most uncertain, so observing it is most informative. The **item information function** of a 2PL item is

$$I_j(\theta) = a_j^2 \, P_j(\theta) \, [1 - P_j(\theta)],$$

which peaks at $\theta = b_j$, where $P_j = 0.5$, with height $a_j^2/4$. A steep item gives a lot of information in a narrow range; a flat item a little information over a wide range. For the 3PL the peak is lower and a little above b_j, because guessing makes correct answers less informative.

Information adds up over items. The **test information function** is the sum of the item information functions, and the **conditional standard error** of a θ estimate is approximately

$$SE(\theta) = \frac{1}{\sqrt{I(\theta)}}.$$

So doubling the information shrinks the standard error by a factor of $\sqrt{2}$, not 2. A test information of 10 gives $SE \approx 0.32$, and on a scale with standard deviation 1 that corresponds to a reliability of about $1 - 0.32^2 = 0.90$ for people at that point.

Two summary reliability coefficients are common:

- **Marginal reliability** averages the error variance over the assumed population distribution of θ: $\rho = 1 - \overline{SE^2}/\sigma^2_\theta$.
- **Empirical reliability** does the same with the estimated scores and their standard errors in the sample.

Both are useful single numbers, but a test can have a respectable overall reliability and still be imprecise exactly where an important decision is made. That is why we look at the curves.

6.2 Example 1: Where does the science test measure well?

```
# SAT12 science items, scored with the corrected key (item 32 keyed 3).
key <- c(1,4,5,2,3,1,2,1,3,1,2,4,2,1,5,3,4,4,1,4,3,3,4,1,3,5,1,3,1,5,4,3)
resp <- key2binary(SAT12, key)
```

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
gg_info(mod) + gg_info(mod, type = "se") + ggplot2::labs(title = "Standard error")
```

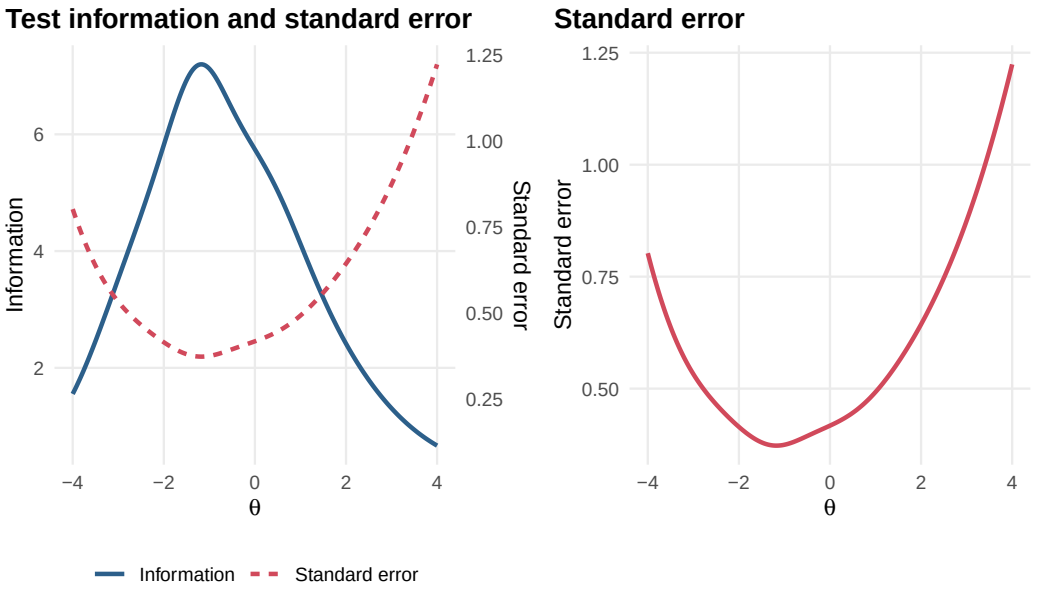


Figure 6.1: Test information (left) and standard error of measurement (right) for the SAT12 science test under the 2PL.

The test is most informative a little below average ability and loses precision quickly above

$\theta = 1$. The values behind the curves:

```
theta <- matrix(-3:3)
data.frame(theta = theta[, 1],
           information = testinfo(mod, theta),
           se = 1 / sqrt(testinfo(mod, theta)))
#>   theta information    se
#> 1    -3       3.53 0.533
#> 2    -2       5.82 0.415
#> 3    -1       7.12 0.375
#> 4     0       5.74 0.417
#> 5     1       4.12 0.492
#> 6     2       2.41 0.644
#> 7     3       1.31 0.875
```

Each item's contribution:

```
gg_info(mod, type = "items", which.items = c(1, 6, 9, 12, 19, 28))
```

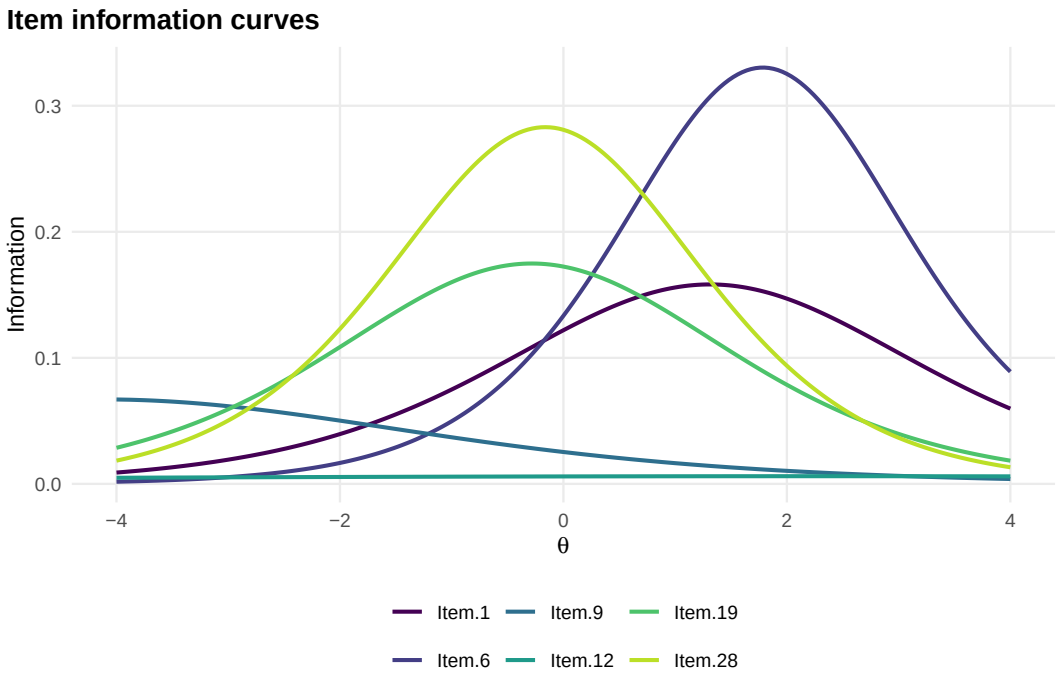


Figure 6.2: Item information curves of six SAT12 items.

Items differ a lot. Steep items in the middle of the scale do most of the work; very easy items (such as Item.9) add information only far below average, and flat items (such as Item.12) add almost nothing anywhere.

Reliability:

```
marginal_rxx(mod)
#> [1] 0.837
empirical_rxx(fscores(mod, full.scores.SE = TRUE))
#>    F1
#> 0.838
```

`areainfo()` tells us how much of the test's total information lies in a range of the scale, here between $\theta = -1$ and $\theta = 1$:

```
areainfo(mod, c(-1, 1))
#>  LowerBound UpperBound Info TotalInfo Proportion nitems
#>          -1          1 11.4      34.2      0.334     32
```

Suppose the school awards a science prize to students above $\theta = 1.5$. At that point the standard error is

```
1 / sqrt(testinfo(mod, matrix(1.5)))
#> [1] 0.559
```

about half a standard deviation. A student estimated at 1.5 could easily have a true ability of 1.0 or 2.0. The overall reliability of about 0.84 gives a much rosier picture than the precision at the decision point.

6.3 Example 2: Designing a test for a decision

Information curves are also a design tool. Imagine a calibrated pool of 90 2PL items for a nursing entrance test, with a pass mark at $\theta = 0.5$. We compare three 20-item forms built from the pool, without collecting any new data: The information function only needs the item parameters.

```
# A pool of 90 calibrated 2PL items (a, b), as stored in an item bank.
# No responses are needed: this example works from item parameters only.
set.seed(606)
pool <- data.frame(item = sprintf("N%02d", 1:90),
                   a = rlnorm(90, 0.15, 0.3),
                   b = rnorm(90, -0.3, 1.1))
```

A small function computes 2PL information from the formula above:

```
info_2pl <- function(items, theta) {
  sapply(theta, function(t) {
    p <- plogis(items$a * (t - items$b))
    sum(items$a^2 * p * (1 - p))
  })
}
```

Three ways to choose 20 items:

```
forms <- list(
  random   = pool[sample(nrow(pool), 20), ],
  spread   = pool[order(pool$b)[round(seq(1, 90, length.out = 20))], ],
  targeted = pool[order(abs(pool$b - 0.5) / pool$a)[1:20], ]
)
grid <- seq(-3, 3, by = 0.05)
curves <- do.call(rbind, lapply(names(forms), function(f)
  data.frame(form = f, theta = grid, info = info_2pl(forms[[f]], grid))))
library(ggplot2)
ggplot(curves, aes(theta, info, colour = form)) +
  geom_line(linewidth = 1) +
  geom_vline(xintercept = 0.5, linetype = 2) +
  labs(x = expression(theta), y = "Test information", colour = "Form") +
  theme_mirtverse()
```

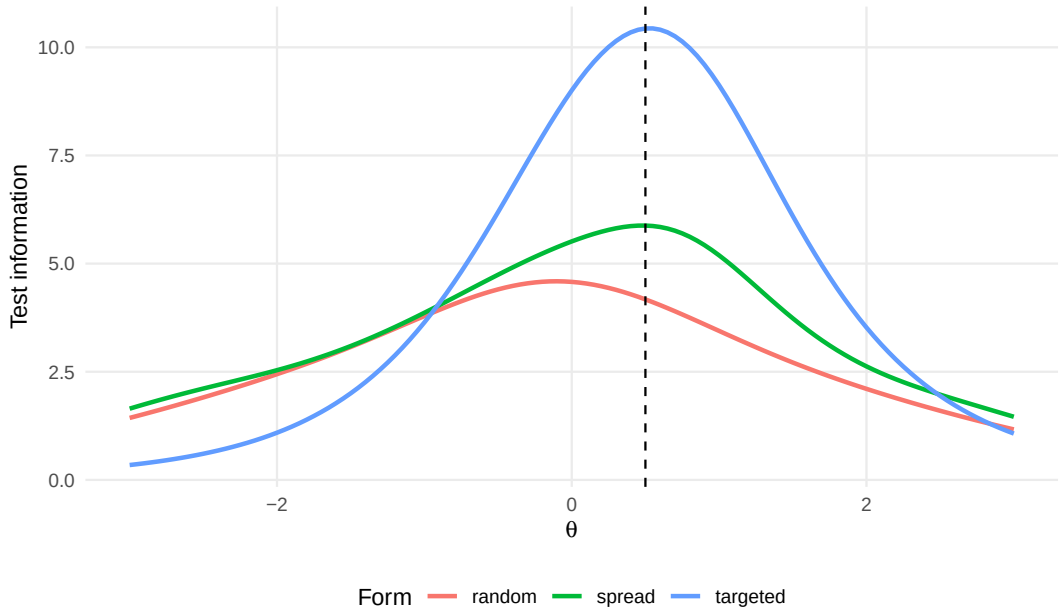


Figure 6.3: Test information of three 20-item forms drawn from the nursing pool; the dashed line marks the pass mark.

The *targeted* form picks items whose difficulty is close to the pass mark, favouring steep ones (dividing by a ranks steep items first). The *spread* form covers the whole range evenly, and the *random* form is what we get by chance. At the pass mark:

```
sapply(forms, function(f) c(info = info_2pl(f, 0.5), se = 1 / sqrt(info_2pl(f,
 ↪  0.5))))
#>      random spread targeted
```

```
#> info   4.17  5.879    10.44
#> se     0.49  0.412     0.31
```

The targeted form has more than twice as much information at the pass mark as the random one, with the same number of items. The price is precision far from the cut: The spread form is better for reporting scores across the whole range. Which is better depends on what the scores are for. A licensing test cares about the pass mark; a progress test cares about everyone. Chapter 11 builds on this idea with automated test assembly and adaptive testing.

6.4 Your turn

1. Show, using `info_2pl()` and a single item with $a = 2$, $b = 0$, that the information peaks at $\theta = 0$ with height $a^2/4 = 1$.
2. In Example 1, how many items like the average SAT12 item would you need to add to bring the standard error at $\theta = 1.5$ down to 0.35?
3. In Example 2, build a 30-item targeted form. How much does the standard error at the pass mark improve over the 20-item targeted form?
4. Explain to a colleague why "the test has a reliability of 0.84" is not enough to decide whether it can be used to award the prize.

Solutions

```
one <- data.frame(a = 2, b = 0)
grid[which.max(info_2pl(one, grid))]; max(info_2pl(one, grid))
#> [1] 0
#> [1] 1

# 2. information needed for SE = 0.35, and the average item information at 1.5
need <- 1 / 0.35^2
have <- testinfo(mod, matrix(1.5))
per_item <- have / 32
ceiling((need - have) / per_item)
#> [1] 50

t30 <- pool[order(abs(pool$b - 0.5) / pool$a)[1:30], ]
1 / sqrt(c(twenty = info_2pl(forms$targeted, 0.5), thirty = info_2pl(t30, 0.5)))
#> twenty thirty
#>  0.310  0.272
```

2. Many more items, because the average item adds little information at 1.5. Adding a few *hard, discriminating* items would be far more efficient than adding average ones.
3. Reliability is an average over everybody. The prize decision happens at $\theta = 1.5$, where the test has much less information than in the middle; what matters is the standard error

there.

In the lab

> The "Precision at the pass mark" section of Lab 2, `run_lab("decisions")`, asks how precisely a certification exam measures candidates close to the pass mark, and what that means for misclassification.

Further reading

Lord (1980) introduced information functions as a tool for test construction; Hambleton, Swaminathan and Rogers (1991) give an accessible account. van der Linden (2005) shows how to build tests to target information curves.

Chapter 7

Scoring people

Learning goals. After this chapter you can

- compare EAP, MAP, ML and WLE scores and say when each is a sensible choice;
- explain why maximum likelihood fails for perfect and zero scores;
- convert sum scores to IRT scores with a conversion table;
- read a Wright map;
- estimate how often people are classified on the wrong side of a cut score.

Key terms: person score, EAP, MAP, ML, WLE, Wright map, classification accuracy.

7.1 The ideas

Once the item parameters are known, we can estimate each person's θ from their responses. The **likelihood** of a response pattern, viewed as a function of θ, is the product of the item probabilities: $P_j(\theta)$ for each item answered correctly and $1 - P_j(\theta)$ for each item answered wrongly. The four common estimators use it differently.

- **ML** (maximum likelihood) takes the θ that maximizes the likelihood. It uses only the responses, which makes it attractive for fairness arguments, but it does not exist for all-correct or all-wrong patterns (the likelihood keeps rising towards $\pm\infty$), and it is biased outwards for short tests.
- **WLE** (weighted likelihood, Warm (1989)) adds a small correction that removes most of ML's bias and gives finite estimates for perfect patterns.
- **MAP** (maximum a posteriori) multiplies the likelihood by a prior distribution for θ, usually the population distribution, and takes the mode of the result.
- **EAP** (expected a posteriori) takes the *mean* of that posterior. It is easy to compute, always finite and has the smallest average error, which is why it is mirt's default. Like MAP, it **shrinks** estimates towards the population mean, more so when the test is short.

The standard error of an EAP estimate is the standard deviation of the posterior, and it comes for free.

For the Rasch model the sum score carries all the information about θ, so everyone with the same sum score gets the same estimate. For the 2PL and 3PL, two people with the same sum score can get different estimates, because correct answers to steep items count for more. Some programmes still want one score per sum score. **EAP for sum scores** (`method = "EAPsum"`) gives exactly that, and produces a conversion table that can be printed in a manual.

7.2 Example 1: Four estimators, one short test

With only five items, the differences between estimators are easy to see.

```
# Five dichotomous LSAT items, one row per test taker.
resp <- expand.table(LSAT7)
```

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
patterns <- unique(resp)
est <- sapply(c("EAP", "MAP", "ML", "WLE"), function(m)
  fscores(mod, method = m, response.pattern = patterns)[, "F1"])
scored <- data.frame(pattern = apply(patterns, 1, paste, collapse = ""),
                     sum = rowSums(patterns), round(est, 2))
scored[order(scored$sum, scored$EAP), ][c(1:4, 15:18, 29:32), ]
#>     pattern sum   EAP   MAP    ML   WLE
#> 1     00000   0 -1.87 -1.82  -Inf -4.14
#> 13    00001   1 -1.53 -1.49 -3.12 -2.60
#> 32    00010   1 -1.51 -1.48 -3.07 -2.56
#> 173   10000   1 -1.41 -1.39 -2.71 -2.31
#> 264   10100   2 -0.65 -0.69 -1.17 -1.15
#> 107   01100   2 -0.61 -0.65 -1.11 -1.10
#> 230   10011   3 -0.75 -0.78 -1.31 -1.26
#> 100   01011   3 -0.70 -0.74 -1.25 -1.21
#> 145   01111   4  0.14  0.06  0.13 -0.22
#> 525   11101   4  0.27  0.18  0.43 -0.03
#> 661   11110   4  0.28  0.20  0.47 -0.01
#> 693   11111   5  0.73  0.64   Inf  1.09
```

Read the table from top to bottom. For the all-wrong pattern ML gives `-Inf`, and for the all-correct pattern `Inf`; WLE, MAP and EAP give finite values. EAP and MAP are pulled towards zero compared with ML and WLE. And patterns with the same sum score get different scores under the 2PL, because Item.3 (the steepest) counts for more than Item.5.

The sum-score conversion table:

```
fscores(mod, method = "EAPsum", full.scores = FALSE)
#>   Sum.Scores     F1 SE_F1 observed expected std.res
#> 0          0 -1.870 0.693       12     10.1   0.601
#> 1          1 -1.432 0.684       40     44.7   0.697
```

```
#> 2          2 -0.949 0.694      114    109.8   0.403
#> 3          3 -0.413 0.721      205    207.7   0.191
#> 4          4  0.152 0.759      321    319.2   0.102
#> 5          5  0.727 0.801      308    308.6   0.031
```

`gg_scores()` shows the distribution of the estimated scores, and `gg_wright()` puts people and items on the same scale. With five easy items the Wright map shows the problem at once: The items sit low on the scale, where few people need to be measured.

```
gg_scores(mod) + gg_wright(mod)
```

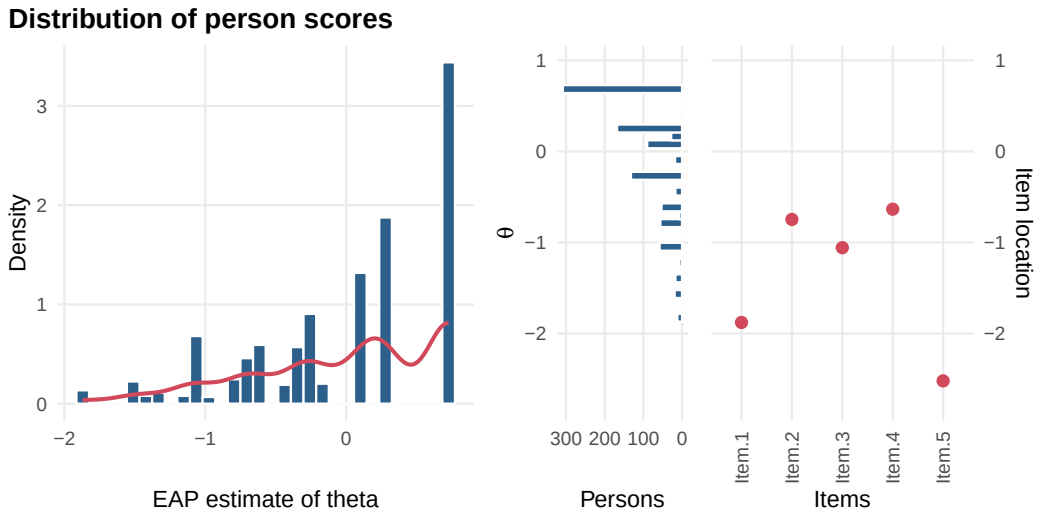


Figure 7.1: Distribution of the estimated scores (left) and the Wright map of people and items (right) for LSAT7.

7.3 Example 2: A proficiency cut on a mathematics test

We simulate a 40-item end-of-year mathematics test for 2,000 students. The province calls students at or above $\theta = 0.8$ *proficient*. Because the data are simulated we know each student's true θ, so we can see how often the estimates put students on the wrong side of the cut.

```
# Simulated 40-item mathematics test; true_theta is known for checking.
set.seed(707)
true_theta <- rnorm(2000)
a <- matrix(rlnorm(40, 0.1, 0.25)); b <- rnorm(40, 0, 1)
resp <- simdata(a, matrix(-a * b), N = 2000, itemtype = "2PL",
                Theta = matrix(true_theta))
colnames(resp) <- sprintf("M%02d", 1:40)
```

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
eap <- fscores(mod, method = "EAP", full.scores.SE = TRUE)
wle <- fscores(mod, method = "WLE", full.scores.SE = TRUE)
c(cor_eap = cor(eap[, 1], true_theta), cor_wle = cor(wle[, 1], true_theta),
  sd_eap = sd(eap[, 1]), sd_wle = sd(wle[, 1]))
#> cor_eap cor_wle  sd_eap  sd_wle
#>   0.945   0.945   0.947   1.049
```

With 40 items both estimators correlate about 0.95 with the true values, and they rank students almost identically. The EAP scores have a smaller spread than the true values (whose standard deviation is 1) and the WLE scores a slightly larger one: Shrinkage on one side, measurement error on the other.

Now the classification. A student is classified correctly if the estimate and the true value fall on the same side of the cut.

```
cut <- 0.8
truth <- true_theta >= cut
observed <- eap[, 1] >= cut
table(true = truth, estimated = observed)
#>        estimated
#> true    FALSE TRUE
#>   FALSE  1503   58
#>   TRUE     81  358
mean(truth == observed)
#> [1] 0.93
```

In practice we never know the truth, but we can estimate the accuracy from the scores and their standard errors (Rudner 2001). If a student's posterior is roughly normal with mean $\hat{\theta}$ and standard deviation $SE(\hat{\theta})$, the probability that their true θ is above the cut is $1 - \Phi\left((c - \hat{\theta})/SE\right)$. Averaging the probability of the side they were classified on gives the expected accuracy:

```
p_above <- pnorm((eap[, 1] - cut) / eap[, 2])
expected_accuracy <- mean(ifelse(observed, p_above, 1 - p_above))
expected_accuracy
#> [1] 0.935
```

The estimate is close to the accuracy we observed against the truth. The misclassified students are, of course, the ones near the cut:

```
library(ggplot2)
ggplot(data.frame(true = true_theta, est = eap[, 1], wrong = truth != observed),
       aes(true, est, colour = wrong)) +
  geom_point(alpha = 0.5, size = 1) +
```

```
  geom_vline(xintercept = cut, linetype = 2) +
  geom_hline(yintercept = cut, linetype = 2) +
  scale_colour_manual(values = c(`FALSE` = "grey60", `TRUE` = "#C0392B"),
                      labels = c("correct", "misclassified")) +
  labs(x = "True theta", y = "EAP estimate", colour = NULL) +
  theme_mirtverse()
```

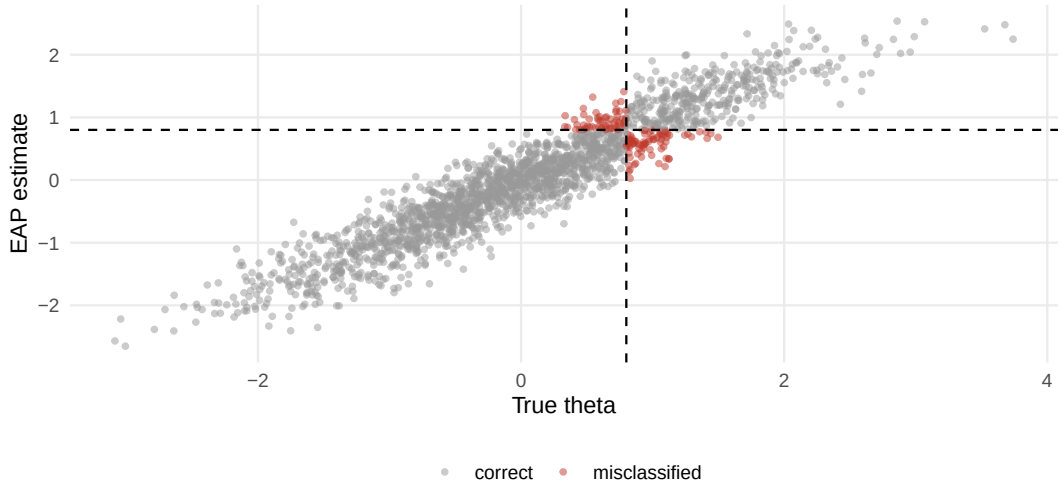


Figure 7.2: True and estimated ability on the mathematics test; misclassified students lie near the cut.

`augment()` returns the scores, their standard errors and the response data as one table, ready to export:

```
head(augment(mod)[, c("M01", "M02", ".F1", ".se_F1")])
#> # A tibble: 6 x 4
#>     M01   M02    .F1 .se_F1
#>   <int> <int>  <dbl>  <dbl>
#> 1     0     0 -1.70   0.384
#> 2     1     0 -0.246  0.304
#> 3     1     0  0.368  0.300
#> 4     1     0  1.18   0.319
#> 5     0     0 -0.746  0.319
#> 6     0     0 -1.09   0.336
```

7.4 Your turn

1. In Example 1, find two response patterns with a sum score of 3 that get different EAP scores. Which items explain the difference?
2. Fit the Rasch model to `LSAT7` and repeat the EAP table. Do patterns with the same sum score still get different scores?
3. In Example 2, compute the expected classification accuracy using only the first 15 items. How much does it drop?
4. A parent asks why their child, who got 30 of 40 right, has a lower score than a classmate who also got 30 right. Write a short answer.

Solutions

```
subset(scored, sum == 3)
#>     pattern sum   EAP   MAP    ML   WLE
#> 65    00111   3 -0.41 -0.47 -0.82 -0.88
#> 100   01011   3 -0.70 -0.74 -1.25 -1.21
#> 114   01101   3 -0.26 -0.32 -0.59 -0.71
#> 137   01110   3 -0.24 -0.31 -0.56 -0.69
#> 230   10011   3 -0.75 -0.78 -1.31 -1.26
#> 278   10101   3 -0.30 -0.37 -0.66 -0.76
#> 329   10110   3 -0.29 -0.35 -0.63 -0.75
#> 440   11001   3 -0.60 -0.64 -1.09 -1.09
#> 465   11010   3 -0.59 -0.63 -1.07 -1.08
#> 507   11100   3 -0.13 -0.20 -0.38 -0.56

rasch <- mirt(expand.table(LSAT7), 1, itemtype = "Rasch", verbose = FALSE)
r_eap <- fscores(rasch, response.pattern = patterns)[, "F1"]
tapply(round(r_eap, 4), rowSums(patterns), function(x) length(unique(x)))
#> 0 1 2 3 4 5
#> 1 1 1 1 1 1

short <- mirt(resp[, 1:15], 1, itemtype = "2PL", verbose = FALSE)
e15 <- fscores(short, full.scores.SE = TRUE)
p15 <- pnorm((e15[, 1] - cut) / e15[, 2])
mean(ifelse(e15[, 1] >= cut, p15, 1 - p15))
#> [1] 0.898
```

2. No: Under the Rasch model every pattern with the same sum score gets the same estimate.
3. For example: “The test gives more weight to questions that are better at telling apart students who have mastered the material from those who have not. Your child and their classmate answered different questions correctly, so their scores differ slightly even though their number correct is the same.”

In the lab

> Lab 2, `run_lab("decisions")`, asks you to choose a scoring method for a certification board, estimate how many candidates are misclassified at the pass mark, and write a memo about it.

Further reading

Embretson and Reise (2000) compare the scoring methods; Warm (1989) introduced the weighted likelihood estimator; Rudner (2001) the approach to expected classification accuracy used here.

Chapter 8

Polytomous items and open-ended responses

> **Learning goals.** After this chapter you can
> - fit the partial credit, generalized partial credit and graded response models, and say how they differ;
> - read category response curves and thresholds;
> - recognise disordered thresholds and decide whether to collapse categories;
> - describe how rater-scored open-ended responses fit into an IRT analysis.

Key terms: polytomous item, graded response model, GPCM, category response curve, threshold, expected score, QWK.

8.1 The ideas

Many items have more than two score categories: Rating scales (*strongly disagree* to *strongly agree*), partial-credit mathematics problems, essays scored 0 to 4 against a rubric. Three families of models are used most.

- The **graded response model** (GRM, Samejima (1969)) models the cumulative probability of scoring in category k *or higher* with a 2PL curve, $P(X \geq k \mid \theta) = \text{logistic}[a(\theta - b_k)]$. The thresholds $b_1 < b_2 < \ldots$ are ordered by construction: b_k is where a person has a 50% chance of scoring k or more.
- The **partial credit model** (PCM, Masters (1982)) is the polytomous Rasch model. It models each *step* from category $k-1$ to k as a Rasch item, with a step difficulty b_k. All items share the same slope.
- The **generalized partial credit model** (GPCM, Muraki (1992)) adds an item slope to the PCM.

In the PCM and GPCM the step difficulties are the points where adjacent category curves cross. They do *not* have to be in order. **Disordered steps** (for example $b_2 < b_1$) mean that a middle

category is never the most likely response at any ability: Respondents skip over it. That is often a sign of a category label that is unclear or a rubric level that raters cannot distinguish, and a common remedy is to merge the category with a neighbour.

Each item's **expected score** curve, the average score as a function of θ, summarizes the categories in one line and is often the easiest plot to explain.

8.2 Example 1: Attitudes to science

`Science` holds the responses of 392 people to four statements about science and technology, each rated on a four-point scale (1 = *strongly disagree* to 4 = *strongly agree*).

```
# Four 4-category attitude items (scored 1-4), 392 respondents.
resp <- Science
```

`select_model()` compares the three models:

```
sel <- select_model(resp, verbose = FALSE)
sel
#> mirtverse model selection (ranked by BIC)
#>
#>        model npar logLik  AIC  BIC SABIC RMSEA   CFI delta weight
#>          GRM   16  -1609 3250 3313  3263 0.148 0.910  0.00  0.795
#>  PCM (Rasch)   13  -1619 3265 3316  3275 0.117 0.860  2.90  0.187
#>         GPCM   16  -1613 3257 3321  3270 0.136 0.924  7.63  0.018
#>
#> The GRM model is recommended because it has the lowest BIC (3313.3). The
#> runner-up, PCM (Rasch), is 2.9 BIC points worse, which is positive evidence for
#> the GRM model. Absolute fit of the GRM model: RMSEA = 0.148 (poor fit), CFI =
#> 0.910, SRMSR = 0.073.
```

The GRM has the lowest BIC, but the absolute fit (RMSEA) is not good for any of the three. With only four items, M2 has few degrees of freedom and RMSEA is unstable, so read it with care, and look at the items.

```
grm <- sel$models[["GRM"]]
tidy(grm)
#> # A tibble: 16 x 5
#>    item    itemtype term  estimate std.error
#>    <chr>   <chr>    <chr>    <dbl>     <dbl>
#>  1 Comfort graded   a         1.04        NA
#>  2 Comfort graded   b1       -4.67        NA
#>  3 Comfort graded   b2       -2.53        NA
#>  4 Comfort graded   b3        1.41        NA
#>  5 Work    graded   a         1.23        NA
```

```
#>  6 Work    graded   b1      -2.39        NA
#>  7 Work    graded   b2      -0.735       NA
#>  8 Work    graded   b3       1.85        NA
#>  9 Future  graded   a        2.29        NA
#> 10 Future  graded   b1      -2.28        NA
#> 11 Future  graded   b2      -0.965       NA
#> 12 Future  graded   b3       0.856       NA
#> 13 Benefit graded   a        1.09        NA
#> 14 Benefit graded   b1      -3.06        NA
#> 15 Benefit graded   b2      -0.906       NA
#> 16 Benefit graded   b3       1.54        NA
```

Each item has one slope and three thresholds. *Future* has by far the steepest slope. The category curves show how the thresholds divide the scale:

```
gg_trace(grm)
```

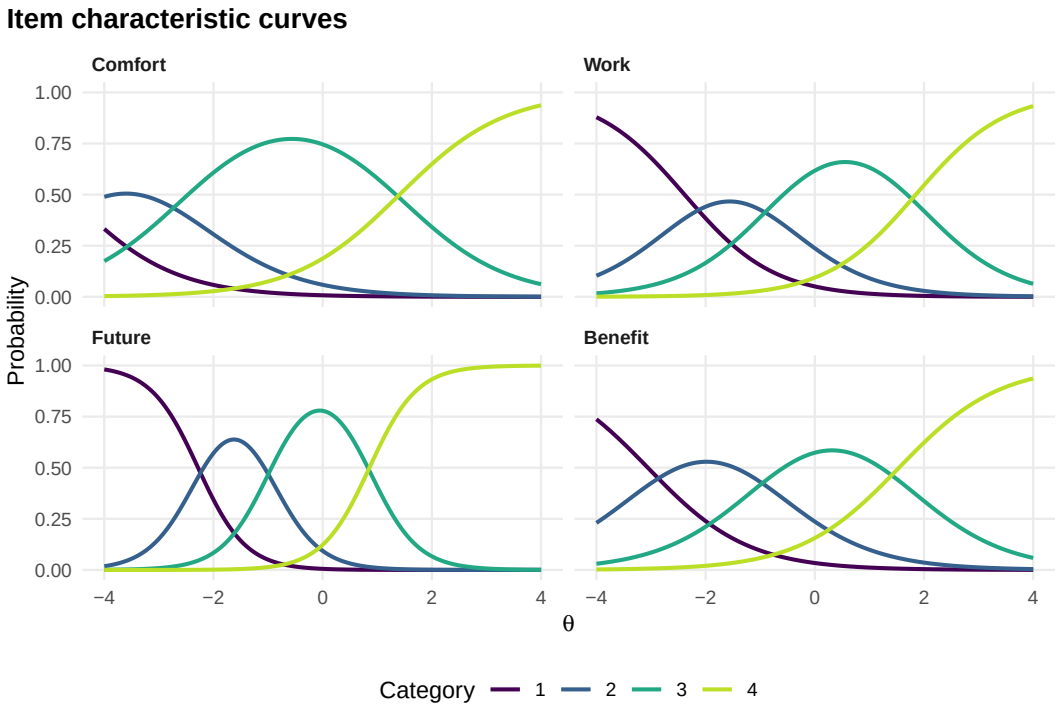


Figure 8.1: Category response curves of the graded response model for the four Science items.

and the expected score curves summarize them:

```
gg_trace(grm, type = "expected", facet = FALSE)
```

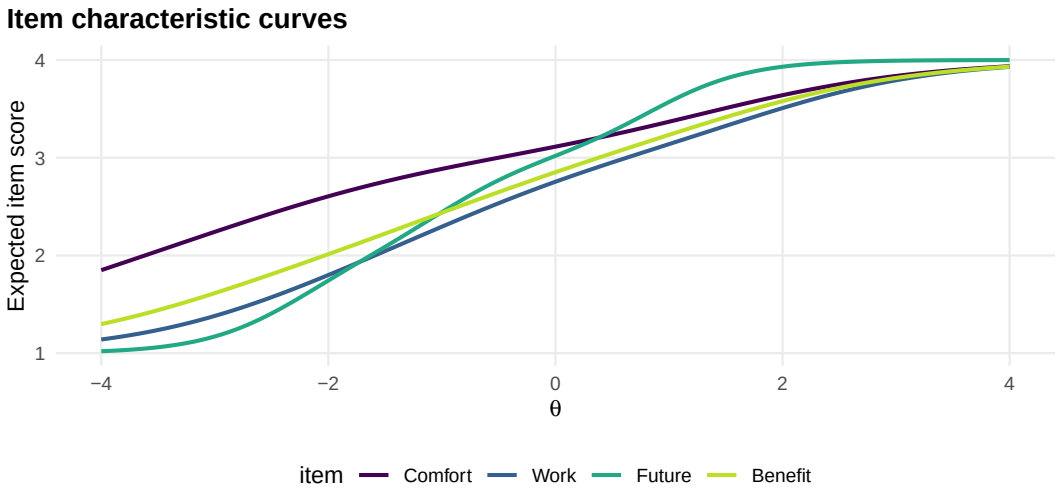


Figure 8.2: Expected item score curves of the graded response model for the Science items.

The lowest category is rare for every item: Its first threshold is far below average, so the curve for "strongly disagree" only rises for people far out in the left tail. The Wright map shows every threshold next to the distribution of people:

```
gg_wright(grm)
```

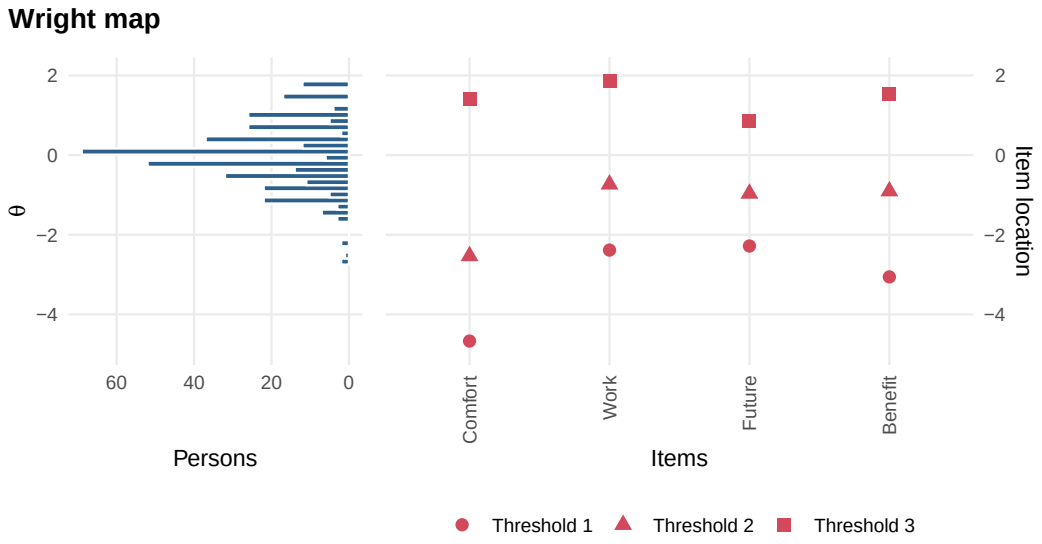


Figure 8.3: Wright map of the Science items: category thresholds next to the distribution of people.

8.3 Example 2: A rating category nobody uses

We simulate a six-item job satisfaction scale with five categories (0 to 4), answered by 1,500 employees. On item J3 the middle label was badly worded, so respondents tend to skip category 2.

```
# Simulated 6-item, 5-category scale; item J3 has disordered steps.
set.seed(808)
steps <- rbind(c(-2.0, -0.8, 0.4, 1.6),
               c(-1.8, -0.6, 0.6, 1.8),
               c(-1.6,  0.9, -0.3, 1.4),   # J3: step 2 harder than step 3
               c(-2.2, -1.0, 0.2, 1.4),
               c(-1.9, -0.7, 0.5, 1.7),
               c(-1.7, -0.5, 0.7, 1.9))
a <- c(1.3, 1.5, 1.4, 1.2, 1.6, 1.3)
d <- cbind(0, -a * t(apply(steps, 1, cumsum)))   # GPCM intercepts
resp <- simdata(matrix(a), d, N = 1500, itemtype = "gpcm")
colnames(resp) <- paste0("J", 1:6)
```

The category frequencies already hint at the problem:

```
round(t(apply(resp, 2, function(x) table(factor(x, levels = 0:4)) / length(x))), 2)
#>       0    1    2    3    4
#> J1 0.05 0.21 0.34 0.28 0.12
#> J2 0.07 0.23 0.37 0.24 0.08
#> J3 0.12 0.41 0.10 0.25 0.13
#> J4 0.05 0.18 0.32 0.30 0.14
#> J5 0.07 0.22 0.36 0.27 0.08
#> J6 0.09 0.24 0.37 0.22 0.08
```

Fit the GPCM and look at the step parameters:

```
gpcm <- mirt(resp, 1, itemtype = "gpcm", verbose = FALSE)
coef(gpcm, IRTpars = TRUE, simplify = TRUE)$items
#>       a    b1     b2     b3   b4
#> J1 1.33 -2.14 -0.831  0.345 1.53
#> J2 1.51 -1.87 -0.665  0.603 1.72
#> J3 1.24 -1.81  1.008 -0.380 1.44
#> J4 1.18 -2.20 -0.963  0.163 1.39
#> J5 1.46 -1.92 -0.723  0.501 1.76
#> J6 1.21 -1.75 -0.625  0.768 1.87
```

For J3 the second step (b_2) is larger than the third (b_3), unlike every other item. The category curves make it plain: Category 2 of J3 is never the most likely response.

```
gg_trace(gpcm, which.items = c("J2", "J3"))
```

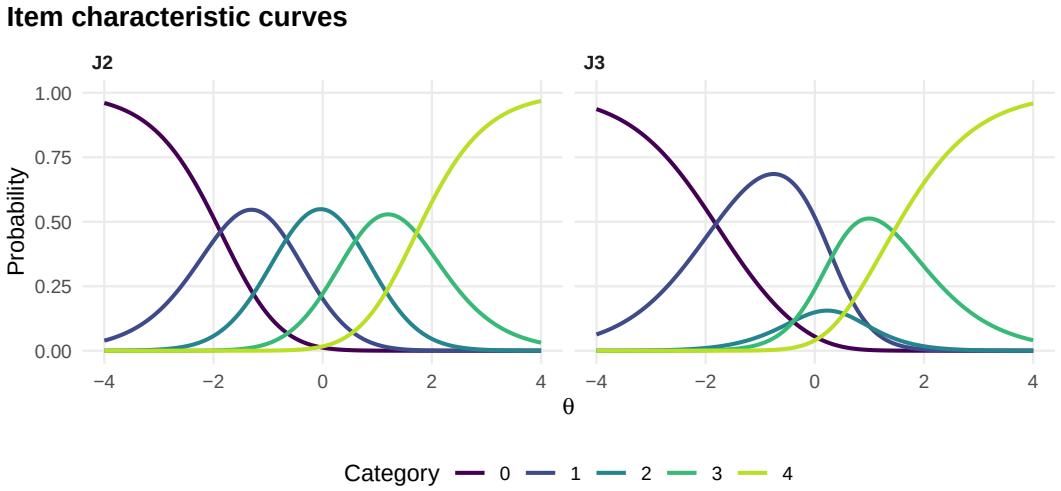


Figure 8.4: Category response curves of J2 and J3; category 2 of J3 is never the most likely response.

Disordered steps are not always a problem; they mean a category is rarely the *most likely* response, not that nobody uses it. Look at the wording or the rubric, and at how many people chose the category. Here we decide that the middle label is unclear and merge categories 1 and 2 of J3, so it is scored 0 to 3:

```
collapsed <- resp
collapsed[, "J3"] <- c(0, 1, 1, 2, 3)[resp[, "J3"] + 1]
gpcm_c <- mirt(collapsed, 1, itemtype = "gpcm", verbose = FALSE)
coef(gpcm_c, IRTpars = TRUE, simplify = TRUE)$items["J3", ]
#>      a     b1     b2     b3     b4
#>  1.613 -1.663  0.575  1.302     NA
```

The steps of the collapsed item are now in order. The GRM would never have shown the problem, because its thresholds are ordered by definition: This is one reason to fit a PCM or GPCM when you are developing a rating scale.

8.4 Open-ended responses and raters

Open-ended tasks, such as essays and constructed responses, are usually scored by trained raters against a rubric, and then analysed with a polytomous model like the ones above. Two extra questions come first. **Do the raters agree?** Agreement on ordinal scores is usually summarized by exact agreement and by **quadratic weighted kappa** (QWK), which gives partial credit for near misses (Cohen 1968). **Are some raters harsher or less consistent than others?**

Rater effects can be studied by comparing each rater's scores with those of the other raters on the same responses, or with models that include rater parameters.

mirtverse's assessment pipeline has a scoring stage for this, `score_open()`, which resolves two reads into a final score and checks agreement (see section 9 of the assessment pipeline article). Lab 6 works through the whole process with eight raters and an automated scoring engine, and Chapter 16 shows how such an engine is trained and evaluated.

8.5 Your turn

1. Fit the PCM (`itemtype = "Rasch"`) to `Science`. Are any of its step difficulties disordered?
2. In Example 2, fit the GRM to the original data. What do the thresholds of J3 look like, and why do they not reveal the problem?
3. Compare the test information of the original and the collapsed GPCM. How much information is lost by collapsing one category of one item?
4. Two raters score 200 essays from 0 to 4 and agree exactly on 60% of them. Why is QWK a better summary than exact agreement for deciding whether the raters can be trusted?

Solutions

```
pcm <- mirt(Science, 1, itemtype = "Rasch", verbose = FALSE)
coef(pcm, IRTpars = TRUE, simplify = TRUE)$items
#>         a    b1     b2   b3
#> Comfort 1 -3.09 -2.595 1.39
#> Work    1 -1.90 -0.911 1.86
#> Future  1 -2.65 -1.421 1.14
#> Benefit 1 -2.45 -0.900 1.36

grm2 <- mirt(resp, 1, itemtype = "graded", verbose = FALSE)
coef(grm2, IRTpars = TRUE, simplify = TRUE)$items["J3", ]
#>       a      b1      b2      b3      b4
#>  1.9899 -1.6164  0.0767  0.4230  1.5352

theta <- matrix(seq(-3, 3, by = 1))
round(rbind(original = testinfo(gpcm, theta), collapsed = testinfo(gpcm_c, theta)),
 ↪  2)
#>           [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> original  1.95 4.19 5.80 6.99 6.16 3.64 1.43
#> collapsed 1.95 4.41 5.89 6.31 6.34 3.85 1.42
```

2. The GRM thresholds of J3 are in order, because the model forces them to be; the second and third are just close together.

3. Hardly any. At some points the collapsed model even has a little more information, because the slope of J3 is estimated higher once the model no longer has to accommodate the skipped category. Collapsing a category that people skip costs almost nothing.
4. Exact agreement treats a 3 versus 4 disagreement the same as a 0 versus 4 disagreement. QWK weights disagreements by their size and corrects for agreement by chance, so it reflects whether the raters rank the essays in the same way.

In the lab

> Lab 6, `run_lab("open-ended")`, follows six rater-scored science tasks from rater agreement to the final IRT model: QWK, rater severity and consistency, resolving two reads, PCM versus GPCM versus GRM, collapsing categories, and checking an automated scoring engine.

Further reading

Samejima (1969), Masters (1982) and Muraki (1992) introduced the three models; Embretson and Reise (2000) compare them. Cohen (1968) introduced weighted kappa.

Chapter 9

Differential item functioning and fairness

Learning goals. After this chapter you can

- tell **impact** (a real difference between groups) from **DIF** (an item that works differently for people of the same ability);
- run an IRT likelihood-ratio DIF analysis and read its effect sizes;
- distinguish uniform from non-uniform DIF;
- run a Mantel-Haenszel DIF test and classify items with the ETS A/B/C rules;
- explain why anchor items matter.

Key terms: DIF, reference and focal groups, uniform and non-uniform DIF, DIF effect size, anchor items.

9.1 The ideas

Groups often differ in their average scores. That alone says nothing about whether a test is fair: The groups may simply differ in what the test measures, which is called **impact**. An item shows **differential item functioning** (DIF) when people from different groups *with the same ability* have different chances of answering it correctly (Holland and Wainer 1993). DIF is a statistical finding, not a verdict of bias; a content review decides whether the reason is irrelevant to what the test should measure (bias) or a real difference in what was taught.

We compare a **reference group** (often the majority group) with a **focal group**. In IRT terms, DIF means the item has different parameters in the two groups.

- **Uniform DIF:** The item is harder for one group at every ability level (the difficulty differs, the curves are parallel).
- **Non-uniform DIF:** The size or direction of the difference changes with ability (the slopes differ and the curves cross).

To compare groups we must first put them on a common scale, and that needs items we trust to work the same way in both: The **anchors**. mirtverse's `dif()` uses a *constrained-baseline* ap-

proach: It fits a multiple-group model in which all item parameters are equal across groups while the group means and variances are free, then frees each item's parameters in turn and tests the change with a likelihood-ratio test. Items you do not test serve as anchors. Because each test involves only one free item, the approach works well when DIF is confined to a few items; when many items have DIF, a *purification* step (testing, removing flagged items from the anchor set, and testing again) is safer.

A significant test is not enough: With large samples tiny differences are significant. `dif()` therefore reports effect sizes on the item score scale. The signed version (`esd`) is the average difference in the probability of a correct answer between the focal and the reference group at the same ability; the unsigned version (`uesd`) also captures crossing curves.

The **Mantel-Haenszel** (MH) procedure (Holland and Thayer 1988) is the classical alternative. It matches people on their sum score, builds a 2×2 table (group × right/wrong) at every score level and tests whether the odds of a correct answer differ between groups across the tables. ETS classifies the size of MH DIF as A (negligible), B (moderate) or C (large).

9.2 Example 1: Paper versus tablet

A district moved its Grade 5 reading test from paper to tablets. We simulate 25 items answered by 1,200 students on paper (reference) and 800 on tablets (focal). The tablet group is slightly weaker on average (impact). Two items that ask students to drag text into a table are harder on tablets (uniform DIF), and one long-passage item is easier for strong tablet readers and harder for weak ones (non-uniform DIF).

```
# Simulated 25-item reading test; `group` is the test mode for each student.
set.seed(909)
a <- matrix(rlnorm(25, 0.2, 0.25)); d <- matrix(rnorm(25, 0.2, 0.9))
a_tab <- a; d_tab <- d
d_tab[c(7, 18)] <- d[c(7, 18)] - 0.8          # R07, R18: uniform DIF
a_tab[22] <- a[22] * 1.8; d_tab[22] <- d[22]   # R22: non-uniform DIF
paper  <- simdata(a, d, N = 1200, itemtype = "2PL")
tablet <- simdata(a_tab, d_tab, N = 800, itemtype = "2PL",
                  Theta = matrix(rnorm(800, -0.2, 1)))
resp <- rbind(paper, tablet)
colnames(resp) <- sprintf("R%02d", 1:25)
group <- factor(rep(c("Paper", "Tablet"), c(1200, 800)),
                levels = c("Paper", "Tablet"))
```

The raw scores differ between the groups, but that alone does not tell us which items, if any, are unfair:

```
mode_resp <- resp; mode_group <- group   # kept for the exercises
tapply(rowSums(resp), group, mean)
#>  Paper Tablet
```

```
#>   12.6   11.1
```

`dif()` tests every item. Paper is the reference group because it is the first level of the factor.

```
res <- dif(resp, group = group)
res
#> DIF analysis: Paper (reference) vs Tablet; 25 items tested
#> Likelihood-ratio tests freeing a1, d; BH-adjusted p < 0.05 flags DIF
#>
#>  item     X2 df p.value p.adjusted flag    esd  uesd converged
#>   R01  1.273  2    .529       .758      -0.007 0.018      TRUE
#>   R02  2.153  2    .341       .758       0.016 0.022      TRUE
#>   R03  0.264  2    .876       .876       0.005 0.009      TRUE
#>   R04  1.480  2    .477       .758      -0.022 0.022      TRUE
#>   R05  1.158  2    .560       .758       0.011 0.019      TRUE
#>   R06  0.953  2    .621       .758      -0.017 0.018      TRUE
#>   R07 29.459  2  < .001     < .001  DIF -0.100 0.100      TRUE
#>   R08  1.447  2    .485       .758       0.021 0.024      TRUE
#>   R09  1.793  2    .408       .758      -0.019 0.021      TRUE
#>   R10  5.327  2    .070       .290       0.013 0.044      TRUE
#>   R11  4.878  2    .087       .312       0.041 0.041      TRUE
#>   R12  7.665  2    .022       .108       0.041 0.047      TRUE
#>   R13  0.720  2    .698       .758       0.001 0.014      TRUE
#>   R14  1.044  2    .593       .758       0.018 0.018      TRUE
#>   R15  0.578  2    .749       .780      -0.007 0.012      TRUE
#>   R16  0.783  2    .676       .758      -0.011 0.013      TRUE
#>   R17  9.572  2    .008       .052       0.061 0.061      TRUE
#>   R18 45.538  2  < .001     < .001  DIF -0.135 0.135      TRUE
#>   R19  2.351  2    .309       .758       0.017 0.024      TRUE
#>   R20  1.087  2    .581       .758      -0.006 0.018      TRUE
#>   R21  1.270  2    .530       .758       0.002 0.019      TRUE
#>   R22 32.166  2  < .001     < .001  DIF  0.043 0.075      TRUE
#>   R23  4.298  2    .117       .364       0.028 0.032      TRUE
#>   R24  0.878  2    .645       .758       0.012 0.015      TRUE
#>   R25  1.456  2    .483       .758       0.024 0.024      TRUE
```

The flagged items are the three we planted. The signed effect sizes of R07 and R18 are negative, meaning the items are harder for tablet users of the same ability. R22's signed effect is small but its unsigned effect is larger: The curves cross, so the advantages and disadvantages partly cancel out on average.

```
gg_dif(res)
```

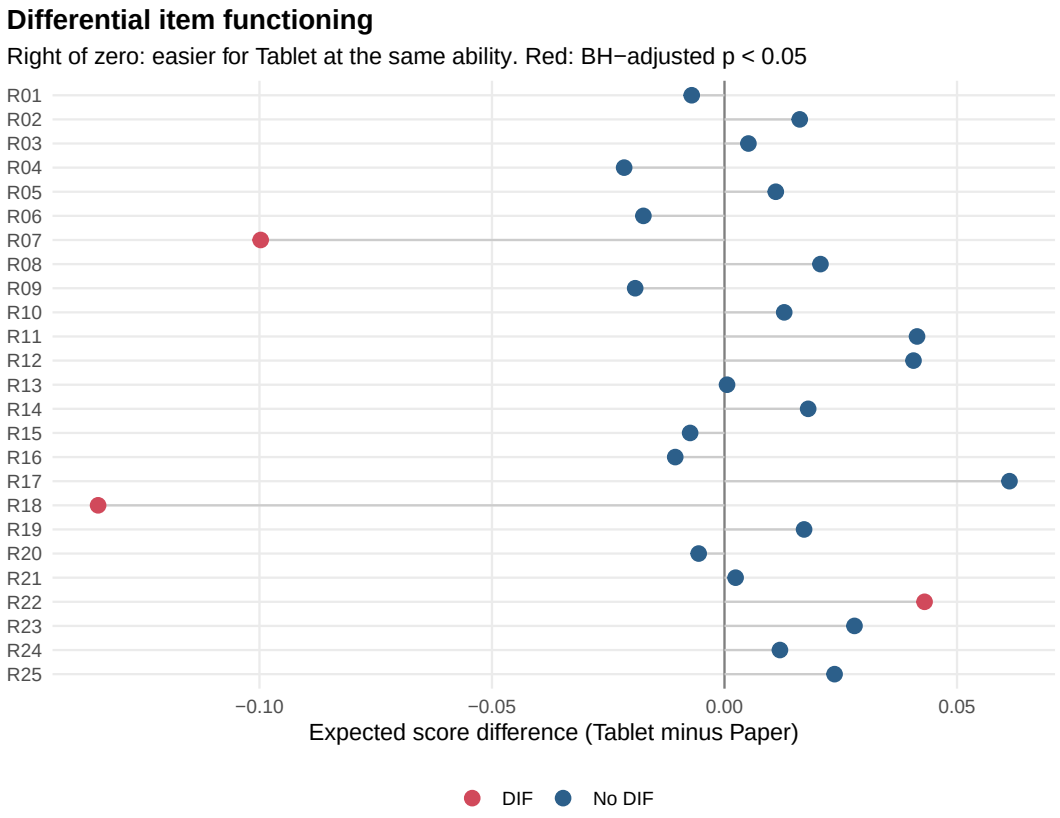


Figure 9.1: DIF effect sizes for the 25 reading items, paper versus tablet.

```
gg_dif(res, type = "curves", which.items = c("R07", "R22"))
```

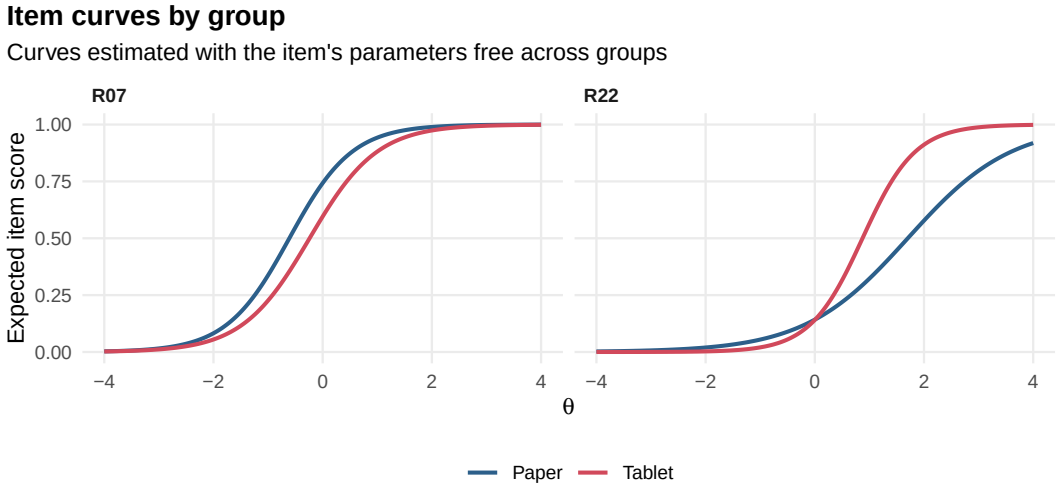


Figure 9.2: Item curves for the paper and tablet groups: uniform DIF in R07, non-uniform DIF in R22.

`interpret()` writes the findings up:

```
interpret(res)
#> DIF overview
#> ------------
#> * Items were tested for DIF between Paper and Tablet, with Paper as the
#>   reference group (n = 1200 Paper, n = 800 Tablet).
#> * Each item's parameters (a1, d) were freed in turn and compared with a model
#>   holding them equal across groups (likelihood-ratio test, BH-adjusted p <
#>   0.05).
#> * On the common scale, the Tablet group's mean ability is 0.28 SD below the
#>   Paper group's, and its SD is 1.09.
#>
#> DIF items
#> ---------
#> * 3 of 25 items show DIF: R18 (expected score difference -0.135, favours
#>   Paper; p < .001); R07 (expected score difference -0.100, favours Paper; p <
#>   .001); R22 (expected score difference 0.043, favours Tablet; p < .001).
#> * Statistical DIF is not proof of bias: ask content experts whether these
#>   items measure something other than the intended ability for one group
#>   before removing or revising them.
```

The baseline model also estimates the impact: The latent mean of the tablet group relative to the paper group (fixed at 0).

```
coef(res$baseline, simplify = TRUE)$Tablet$means
#>     F1
#> -0.283
```

9.2.1 Choosing anchors

If we already know some items are clean, we can test only the others: The untested items become anchors. Here, as a check, we test only the three flagged items against the other 22:

```
dif(resp, group = group, items = c("R07", "R18", "R22"))$table
#> # A tibble: 3 x 9
#>   item     X2    df  p.value p.adjusted flag      esd   uesd converged
#>   <chr> <dbl> <int>    <dbl>      <dbl> <lgl>   <dbl>  <dbl> <lgl>
#> 1 R07    29.5     2 4.01e- 7   4.01e- 7 TRUE  -0.0997 0.0997 TRUE
#> 2 R18    45.5     2 1.29e-10   3.88e-10 TRUE  -0.135  0.135  TRUE
#> 3 R22    32.2     2 1.04e- 7   1.55e- 7 TRUE   0.0430 0.0755 TRUE
```

Does the DIF matter for scores? One way to judge is to compare the expected total score of the test in the two groups at the same ability, which is called differential test functioning. With two items that are harder for tablet users and one that crosses, the total effect is a fraction of a

score point; a content review would still look at the drag-and-drop items, because the problem can be fixed by changing the interface.

9.3 Example 2: Mantel-Haenszel on a chemistry quiz

The MH test needs no IRT model, which makes it a good first screen and a useful check. We simulate a 20-item chemistry quiz taken by 600 girls and 600 boys of equal average ability; item C11 involves a sports context and is easier for boys of the same ability.

```
# Simulated 20-item quiz; `group` is gender. C11 has uniform DIF.
set.seed(910)
a <- matrix(rlnorm(20, 0.2, 0.2)); d <- matrix(rnorm(20, 0, 0.8))
d_boys <- d; d_boys[11] <- d[11] + 0.7
resp <- rbind(simdata(a, d, N = 600, itemtype = "2PL"),
              simdata(a, d_boys, N = 600, itemtype = "2PL"))
colnames(resp) <- sprintf("C%02d", 1:20)
group <- factor(rep(c("Girl", "Boy"), each = 600), levels = c("Girl", "Boy"))
```

For each item we match on the total score of the *other* items, build the stratified 2×2 tables and run the MH test. The common odds ratio α_{MH} converts to the ETS delta scale as MH D-DIF $= -2.35 \ln \alpha_{MH}$. The ETS rules call DIF negligible (A) when $|\text{MH D-DIF}| < 1$ or the test is not significant, large (C) when $|\text{MH D-DIF}| \geq 1.5$ and significant, and moderate (B) otherwise (the full rules also test the size against 1).

```
mh_item <- function(resp, group, item) {
  rest <- rowSums(resp[, colnames(resp) != item])
  strata <- cut(rest, unique(quantile(rest, 0:5 / 5)), include.lowest = TRUE)
  tab <- table(group, factor(resp[, item], levels = 1:0), strata)
  mh <- mantelhaen.test(tab, exact = FALSE)
  delta <- -2.35 * log(unname(mh$estimate))
  data.frame(item = item, alpha_MH = unname(mh$estimate),
             MH_D_DIF = delta, p.value = mh$p.value,
             ETS = ifelse(mh$p.value >= 0.05 | abs(delta) < 1, "A",
                          ifelse(abs(delta) >= 1.5, "C", "B")))
}
mh <- do.call(rbind, lapply(colnames(resp), mh_item, resp = resp, group = group))
mh[order(mh$p.value), ][1:5, ]
#>    item alpha_MH MH_D_DIF  p.value ETS
#> 11  C11    0.576    1.297 3.55e-05   B
#> 4   C04    1.288   -0.596 4.54e-02   A
#> 13  C13    0.789    0.556 1.22e-01   A
#> 1   C01    1.222   -0.471 1.36e-01   A
#> 9   C09    1.224   -0.476 1.42e-01   A
```

The table is arranged so that α_{MH} is the odds of a correct answer in the reference group (girls)

divided by the odds in the focal group (boys), at the same score. Values above 1, and negative MH D-DIF, mean the item favours the reference group; values below 1, and positive MH D-DIF, favour the focal group. C11 stands out, with a moderate (B) DIF in favour of boys. Matching on five score bands keeps the tables reasonably full in a sample of this size; with more people you can match on every score.

The IRT analysis agrees:

```
subset(dif(resp, group = group)$table, flag)
#> # A tibble: 1 x 9
#>   item     X2    df  p.value p.adjusted flag    esd  uesd converged
#>   <chr> <dbl> <int>    <dbl>      <dbl> <lgl> <dbl> <dbl> <lgl>
#> 1 C11    18.1     2 0.000117    0.00233 TRUE  0.104 0.104 TRUE
```

9.4 Your turn

1. In Example 1, refit `dif()` with `p_adjust = "none"`. How many items are flagged now? Which result would you report, and why?
2. Run the MH procedure on the data of Example 1. Does it detect R22? Why is MH weaker for non-uniform DIF?
3. In Example 2, what happens to the MH results if you match on the total score *including* the studied item? On five versus ten score bands?
4. A newspaper writes that "the tablet test is biased because tablet users scored lower". Using the results of Example 1, write three sentences in reply.

Solutions

```
sum(dif(mode_resp, group = mode_group, p_adjust = "none")$table$flag)
#> [1] 5
mh_mode <- do.call(rbind, lapply(colnames(mode_resp), mh_item,
                                 resp = mode_resp, group = mode_group))
subset(mh_mode, item %in% c("R07", "R18", "R22"))
#>    item alpha_MH MH_D_DIF  p.value ETS
#> 7   R07    1.980   -1.605 1.16e-09   C
#> 18  R18    2.079   -1.720 1.57e-12   C
#> 22  R22    0.689    0.876 4.67e-03   A
```

1. Without adjustment a few more items may be flagged at the 5% level by chance alone, since 25 tests are run. Report the adjusted results, and use the effect sizes to decide which items matter.
2. MH compares the *average* odds across score levels. When the curves cross, the advantage at one end and the disadvantage at the other cancel out, so MH (like the signed effect size) often misses non-uniform DIF. The IRT likelihood-ratio test, which frees the slope, catches it.

3. Including the studied item in the matching score is the textbook MH procedure; excluding it, as above, is a common variant. With 20 items the results are similar. More bands make the matching finer, but leave fewer people per table.
4. For example: "Tablet users did score a little lower on average, but most of that difference reflects a real difference in reading between the groups, not the test. When we compared students of the same reading ability, only three of 25 questions worked differently, and two of them involve dragging text on a screen, which we are redesigning."

In the lab

Lab 3, `run_lab("fairness")`, investigates a complaint that a numeracy test is biased against English language learners, with Mantel-Haenszel and IRT DIF tests, anchor purification and differential test functioning. Lab 2 has a shorter DIF section on a certification exam.

Further reading

Holland and Wainer (1993) remain the standard reference; Holland and Thayer (1988) introduced the MH procedure for DIF.

Chapter 10

Linking and equating

Learning goals. After this chapter you can

- explain why two separately calibrated forms are not on the same scale;
- describe common-item (anchor) and common-person designs;
- link two calibrations with the mean-sigma and Stocking-Lord methods;
- check anchor items for parameter drift;
- run a concurrent calibration with a multiple-group model;
- carry a cut score from one form to another with IRT true-score equating.

Key terms: linking, anchor items, moment linking, characteristic curve linking, parameter drift, test characteristic curve.

10.1 The ideas

Every calibration fixes the scale by assuming that the people in it have θ with mean 0 and standard deviation 1. If this year's candidates are stronger than last year's, a separate calibration of this year's form will still put *them* at mean 0, and every item will look harder than it is. To compare forms, or groups, the two calibrations must be put on a common scale. That is **linking**; when the goal is to make scores on two forms interchangeable, it is called **equating** (Kolen and Brennan 2014).

Because the IRT scale is only determined up to a linear transformation, linking amounts to finding two constants A and B such that, for the new form's parameters,

$$\theta^* = A\theta + B, \qquad b^* = Ab + B, \qquad a^* = a/A.$$

The constants come from information the two calibrations share. In the common design, the **non-equivalent groups with anchor test** (NEAT) design, the two forms share a set of **anchor items**. Two families of methods estimate A and B from the anchors.

- **Moment methods.** *Mean-sigma* matches the mean and standard deviation of the anchor difficulties: $A = s_{b,\text{old}}/s_{b,\text{new}}$ and $B = \bar{b}_{\text{old}} - A\,\bar{b}_{\text{new}}$. They are simple, but sensitive to a single unusual anchor.
- **Characteristic curve methods.** *Stocking-Lord* (Stocking and Lord 1983) chooses A and B to make the anchors' test characteristic curve (the expected anchor score as a function of θ) as similar as possible on the two scales. They use all the parameters at once and are more robust.

An alternative to linking two separate calibrations is **concurrent calibration**: Fit one multiple-group model to both forms, constraining the anchor items to be equal across groups and estimating the new group's mean and standard deviation.

Everything rests on the anchors behaving the same way in both administrations. An anchor whose parameters have changed, because it was exposed, taught to, or its position or wording changed, shows **parameter drift**, and it must be removed from the anchor set before linking.

10.2 Example 1: Separate calibration and linking

A language school has an old and a new 30-item placement test that share 10 anchor items. We simulate 1,000 students on each. The new cohort is stronger (mean 0.4, standard deviation 1.1), and anchor A06 was leaked and became easier on the new form.

```
# Simulated NEAT design: old form (O01-O20 + anchors A01-A10) and new form
# (N01-N20 + the same anchors); anchor A06 drifts on the new form.
set.seed(1010)
anchors <- sprintf("A%02d", 1:10)
pars <- data.frame(item = c(sprintf("O%02d", 1:20), sprintf("N%02d", 1:20),
 ↪  anchors),
                   a = rlnorm(50, 0.15, 0.25), b = rnorm(50, 0, 1))
gen <- function(items, n, mean, sd, shift = NULL) {
  p <- pars[match(items, pars$item), ]
  if (!is.null(shift)) p$b[p$item %in% names(shift)] <- p$b[p$item %in%
   ↪  names(shift)] + shift
  out <- simdata(matrix(p$a), matrix(-p$a * p$b), N = n, itemtype = "2PL",
                 Theta = matrix(rnorm(n, mean, sd)))
  colnames(out) <- items
  as.data.frame(out)
}
old <- gen(c(sprintf("O%02d", 1:20), anchors), 1000, 0, 1)
new <- gen(c(sprintf("N%02d", 1:20), anchors), 1000, 0.4, 1.1, shift = c(A06 =
 ↪  -1.2))
```

Calibrate each form on its own:

```
fit_old <- mirt(old, 1, itemtype = "2PL", verbose = FALSE)
fit_new <- mirt(new, 1, itemtype = "2PL", verbose = FALSE)
p_old <- coef(fit_old, IRTpars = TRUE, simplify = TRUE)$items[anchors, c("a", "b")]
p_new <- coef(fit_new, IRTpars = TRUE, simplify = TRUE)$items[anchors, c("a", "b")]
round(cbind(b_old = p_old[, "b"], b_new = p_new[, "b"]), 2)
#>     b_old b_new
#> A01 -0.49 -0.90
#> A02  0.69  0.18
#> A03 -1.44 -1.74
#> A04 -0.13 -0.38
#> A05 -1.05 -1.53
#> A06 -0.43 -2.10
#> A07 -1.10 -1.51
#> A08 -0.62 -1.02
#> A09  1.14  0.65
#> A10 -1.82 -2.15
```

The anchors look easier on the new calibration, because the new students are stronger and their calibration puts them at mean 0. Before linking, check for drift. On a common scale the anchor difficulties should lie on a straight line:

```
library(ggplot2)
ggplot(data.frame(old = p_old[, "b"], new = p_new[, "b"], item = anchors),
       aes(new, old)) +
  geom_smooth(method = "lm", se = FALSE, colour = "grey70", formula = y ~ x) +
  geom_point() + geom_text(aes(label = item), nudge_y = 0.15, size = 3) +
  labs(x = "Difficulty, new calibration", y = "Difficulty, old calibration") +
  theme_mirtverse()
```

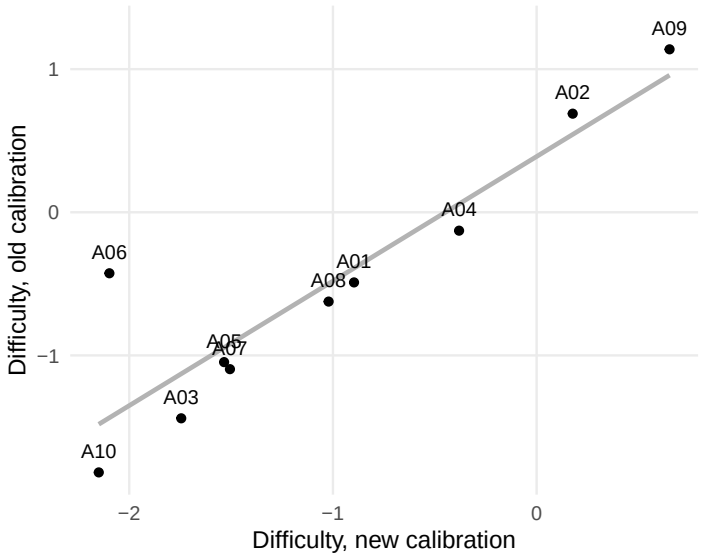


Figure 10.1: Anchor difficulties in the old and new calibrations; items off the line may have drifted.

A06 sits well off the line formed by the others. A simple rule is to flag anchors whose difference in difficulty is far from the typical difference:

```
diff_b <- p_old[, "b"] - p_new[, "b"]
round(sort((diff_b - median(diff_b)) / mad(diff_b)), 1)
#>  A04  A03  A10  A08  A01  A07  A09  A05  A02  A06
#> -1.3 -0.9 -0.7 -0.1  0.0  0.0  0.7  0.7  0.9 10.8
```

We drop A06 from the anchor set and compute the mean-sigma constants from the remaining nine:

```
keep <- setdiff(anchors, "A06")
A <- sd(p_old[keep, "b"]) / sd(p_new[keep, "b"])
B <- mean(p_old[keep, "b"]) - A * mean(p_new[keep, "b"])
c(A = A, B = B)
#>    A    B
#> 1.04 0.44
```

B estimates how much stronger the new cohort is on the old scale (we simulated 0.4), and A how much more spread out (1.1). Stocking-Lord finds the constants that make the two anchor characteristic curves agree:

```
tcc <- function(a, b, theta) sapply(theta, function(t) sum(plogis(a * (t - b))))
grid <- seq(-3, 3, length.out = 41)
sl_loss <- function(k) {
  A <- k[1]; B <- k[2]
```

```
  sum((tcc(p_old[keep, "a"], p_old[keep, "b"], grid) -
       tcc(p_new[keep, "a"] / A, A * p_new[keep, "b"] + B, grid))^2 * dnorm(grid))
}
sl <- optim(c(1, 0), sl_loss)$par
c(A = sl[1], B = sl[2])
#>    A    B
#> 1.05 0.41
```

Both methods recover the cohort difference well. What would have happened if we had kept the drifted anchor?

```
A_all <- sd(p_old[, "b"]) / sd(p_new[, "b"])
c(A = A_all, B = mean(p_old[, "b"]) - A_all * mean(p_new[, "b"]))
#>     A     B
#> 0.963 0.487
```

One leaked item is enough to distort the link, and with it every score on the new form.

10.3 Example 2: Concurrent calibration and equating a pass mark

Concurrent calibration fits both forms at once. We stack the two datasets; each student has `NA` for the items they did not take.

```
# The same NEAT data stacked: NA for items not on a student's form.
# A06 is renamed on the new form, so it is not treated as an anchor.
new_a <- new; names(new_a)[names(new_a) == "A06"] <- "A06_new"
resp <- merge(transform(old, form = "old", id = seq_len(nrow(old))),
              transform(new_a, form = "new", id = nrow(old) + seq_len(nrow(new_a))),
              all = TRUE)
form <- factor(resp$form, levels = c("old", "new"))
resp <- resp[, setdiff(names(resp), c("form", "id"))]
```

Renaming A06 on the new form turns it into two separate items, one per form, which is the usual way to drop a drifted anchor from the link without discarding its responses. `multipleGroup()` constrains every item to have the same parameters in both groups (each unique item is only seen by one group anyway) and frees the new group's mean and variance:

```
mg <- suppressWarnings(
  multipleGroup(resp, 1, group = form, itemtype = "2PL", verbose = FALSE,
                invariance = c(colnames(resp), "free_means", "free_var")))
coef(mg, simplify = TRUE)$new[c("means", "cov")]
#> $means
#>    F1
#> 0.411
```

```
#>
#> $cov
#>      F1
#> F1 1.12
```

The new group's mean and variance are estimated on the old group's scale, and are close to the values we simulated (0.4 and $1.1^2 = 1.21$).

Now the pass mark. The old form's pass mark was 18 correct out of 30. What score on the new form corresponds to the same ability? **IRT true-score equating** finds the θ at which the old form's expected score (its test characteristic curve) equals 18, then reads off the new form's expected score at that θ.

```
items <- coef(mg, IRTpars = TRUE, simplify = TRUE)$old$items
old_items <- c(sprintf("O%02d", 1:20), anchors)
new_items <- c(sprintf("N%02d", 1:20), setdiff(anchors, "A06"), "A06_new")
tcc_form <- function(ids, theta) tcc(items[ids, "a"], items[ids, "b"], theta)
theta_cut <- uniroot(function(t) tcc_form(old_items, t) - 18, c(-4, 4))$root
c(theta_cut = theta_cut, new_form_cut = tcc_form(new_items, theta_cut))
#>    theta_cut new_form_cut
#>        0.419       19.763
```

```
curves <- data.frame(theta = rep(grid, 2),
                     score = c(tcc_form(old_items, grid), tcc_form(new_items, grid)),
                       form = rep(c("Old form", "New form"), each = length(grid)))
ggplot(curves, aes(theta, score, colour = form)) +
  geom_line(linewidth = 1) +
  geom_vline(xintercept = theta_cut, linetype = 2) +
  geom_hline(yintercept = 18, linetype = 3) +
  labs(x = expression(theta), y = "Expected number correct", colour = NULL) +
  theme_mirtverse()
```

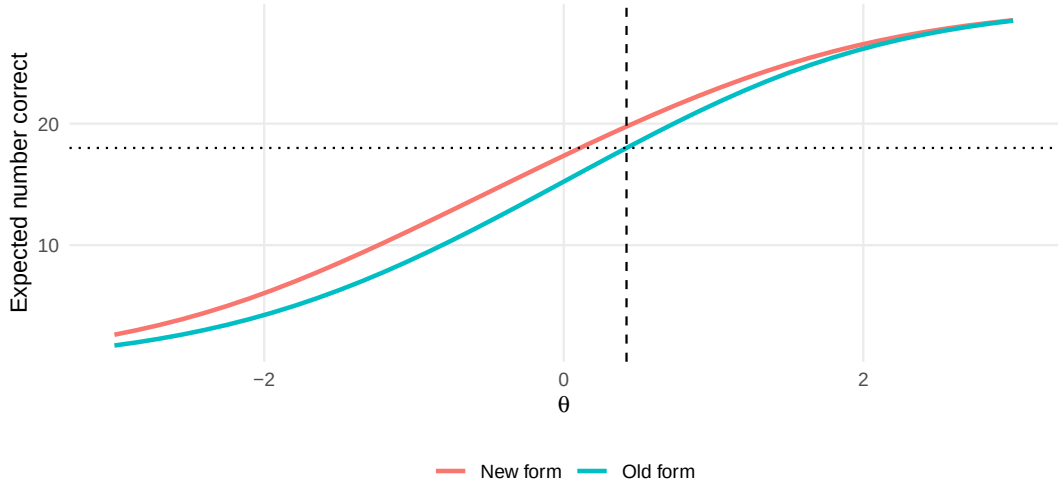


Figure 10.2: Test characteristic curves of the old and new forms on the common scale, with the cut score.

If the new form is easier, its curve lies above the old form's and the pass mark on the new form is higher; if it is harder, the pass mark is lower. Here the new form is a little easier, so the old pass mark of 18 corresponds to about 19.8 on the new form. Rounding the equated score to a whole number is a policy decision: Rounding down passes slightly more candidates than the old standard, rounding up slightly fewer.

10.4 Your turn

1. Repeat the mean-sigma link in Example 1 with the *mean-mean* method: $A = \bar{a}_{\text{new}}/\bar{a}_{\text{old}}$, $B = \bar{b}_{\text{old}} - A\,\bar{b}_{\text{new}}$. How close are the constants to the Stocking-Lord ones?
2. Rerun the concurrent calibration of Example 2 *with* A06 as an anchor. How much does the estimated mean of the new group change?
3. Compute the pass mark on the new form if the old pass mark were 22.
4. A board member asks why the pass mark changed from one year to the next "when the standard was supposed to stay the same". Answer in two sentences.

Solutions

```
A_mm <- mean(p_new[keep, "a"]) / mean(p_old[keep, "a"])
c(A = A_mm, B = mean(p_old[keep, "b"]) - A_mm * mean(p_new[keep, "b"]))
#>     A     B
#> 1.025 0.422

resp_all <- merge(transform(old, form = "old", id = seq_len(nrow(old))),
                  transform(new, form = "new", id = nrow(old) + seq_len(nrow(new))),
```

```
                  all = TRUE)
form_all <- factor(resp_all$form, levels = c("old", "new"))
resp_all <- resp_all[, setdiff(names(resp_all), c("form", "id"))]
mg_all <- suppressWarnings(
  multipleGroup(resp_all, 1, group = form_all, itemtype = "2PL", verbose = FALSE,
                invariance = c(colnames(resp_all), "free_means", "free_var")))
coef(mg_all, simplify = TRUE)$new$means
#>   F1
#> 0.54

theta22 <- uniroot(function(t) tcc_form(old_items, t) - 22, c(-4, 4))$root
tcc_form(new_items, theta22)
#> [1] 23.1
```

2. With A06 treated as an anchor, the new group's mean rises well above the simulated 0.4. The model has to explain why the new students do so much better on A06; since A06 is constrained to be the same item, part of the leak is credited to the students' ability. Every new-form score would be too high.
3. For example: "The standard is the same level of ability, not the same number of questions. This year's form was a little easier, so reaching that level of ability takes a slightly higher number of correct answers."

In the lab

> Lab 4, `run_lab("linking")`, asks why a licensure exam's pass rate jumped on a new form. You calibrate both forms, link them, check the anchors, compare separate and concurrent calibration, and carry the pass mark over.

Further reading

Kolen and Brennan (2014) is the standard reference on equating and linking. Stocking and Lord (1983) introduced the characteristic curve method used here.

Chapter 11

Item banks, test assembly and adaptive testing

Learning goals. After this chapter you can

- describe what an item bank stores and what it is for;
- read a bank's information curve and spot where the bank is thin;
- assemble a fixed form from a bank under content constraints;
- explain, step by step, how a computerized adaptive test (CAT) selects items, scores people and decides when to stop;
- judge a CAT design by its precision, test length and item exposure.

Key terms: item bank, automated test assembly, parallel forms, computerized adaptive test, multistage test, exposure.

11.1 The ideas

Once items have been calibrated on a common scale (Chapter 10), their parameters can be stored in an **item bank** together with their content codes, usage history and review status. Because the parameters are on one scale, any selection of items from the bank yields scores on that scale. That makes two things possible.

Automated test assembly (ATA) picks items for a fixed form so that the form meets a blueprint (for example, ten items from each content area) and its information curve matches a target: High at a cut score for a licensing test, flat across the range for a progress test (Linden 2005). Exact solutions use mixed-integer programming; simple greedy rules, which add the best remaining item that still fits the blueprint, often get close.

A **computerized adaptive test** (CAT) goes further and builds a different test for every person while they take it (Wainer et al. 2000). The loop is:

1. Start with a provisional θ (often 0).

2. Choose the unused item with the most information at the current θ, subject to content and exposure rules.
3. Score the response and update θ (usually by EAP, which works even after one item).
4. Stop when the standard error is small enough or a maximum length is reached; otherwise go back to step 2.

A CAT typically reaches the precision of a fixed test with half as many items or fewer. The price is a bank large enough to cover the whole range, and **exposure**: Without control, the most informative items are given to almost everybody and soon become known. A **multistage test** (MST) is a compromise: People take pre-assembled modules, and the next module depends on their performance on the last one.

11.2 Example 1: A vocabulary bank and a fixed form

A language-assessment centre has a calibrated bank of 200 vocabulary items in four areas.

```
# A calibrated 2PL item bank: 200 items, four content areas.
set.seed(1111)
bank <- data.frame(item = sprintf("V%03d", 1:200),
                   area = factor(rep(c("Everyday", "Academic", "Workplace",
                    ↪  "Idioms"),
                                     each = 50)),
                   a = rlnorm(200, 0.2, 0.3),
                   b = c(rnorm(50, -0.8, 0.9), rnorm(50, 0.4, 0.9),
                         rnorm(50, 0, 0.9),    rnorm(50, 0.9, 0.7)))
```

The bank's information curve, overall and by area, shows where it can measure well:

```
item_info <- function(a, b, theta) {
  p <- plogis(a * (theta - b)); a^2 * p * (1 - p)
}
grid <- seq(-3, 3, by = 0.1)
bank_info <- do.call(rbind, lapply(levels(bank$area), function(ar) {
  s <- bank[bank$area == ar, ]
  data.frame(area = ar, theta = grid,
             info = sapply(grid, function(t) sum(item_info(s$a, s$b, t))))
}))
library(ggplot2)
ggplot(bank_info, aes(theta, info, fill = area)) +
  geom_area() +
  labs(x = expression(theta), y = "Bank information", fill = NULL) +
  theme_mirtverse()
```

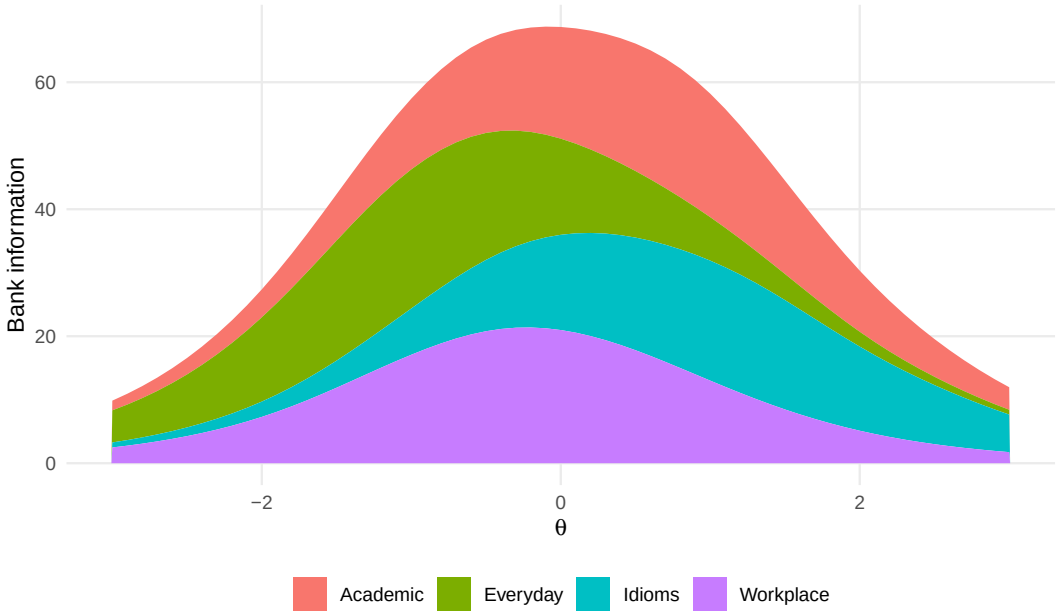


Figure 11.1: Information of the 200-item vocabulary bank, stacked by content area.

The bank is rich in the middle and thin at the extremes, especially below $\theta = -2$ and above $\theta = 2$. Everyday vocabulary covers the low end, idioms the high end. Item writers should be asked for very easy and very hard items.

Now a 24-item fixed form with six items from each area, to measure well between $\theta = -1$ and $\theta = 1$. A greedy assembly adds, one at a time, the item that adds the most information at the target points, as long as its area is not yet full:

```
assemble_greedy <- function(bank, length, per_area, targets) {
  chosen <- integer(0)
  while (length(chosen) < length) {
    full <- names(which(table(bank$area[chosen]) >= per_area))
    cand <- setdiff(which(!bank$area %in% full), chosen)
    gain <- sapply(cand, function(i) sum(item_info(bank$a[i], bank$b[i], targets)))
    chosen <- c(chosen, cand[which.max(gain)])
  }
  bank[chosen, ]
}
form <- assemble_greedy(bank, length = 24, per_area = 6, targets = c(-1, 0, 1))
table(form$area)
#>
#>  Academic  Everyday    Idioms Workplace
#>         6         6         6         6
```

Compared with a random 24-item form that meets the same blueprint:

```
random_form <- do.call(rbind, lapply(split(bank, bank$area), function(s)
 ↪  s[sample(50, 6), ]))
se_at <- function(f, theta) 1 / sqrt(sapply(theta, function(t) sum(item_info(f$a,
 ↪  f$b, t))))
round(rbind(assembled = se_at(form, c(-1, 0, 1)),
            random = se_at(random_form, c(-1, 0, 1))), 3)
#>            [,1]  [,2]  [,3]
#> assembled 0.265 0.236 0.276
#> random    0.318 0.304 0.332
```

The assembled form measures more precisely at all three target points with the same length and blueprint. For real programmes, `assemble_forms()` in the mirtverse assessment pipeline solves the same problem exactly, builds several parallel forms at once and handles enemy items (items that must not appear together).

11.3 Example 2: A CAT from scratch

We now write a complete CAT in about 30 lines, using the same bank. The scoring step computes the EAP estimate on a grid of quadrature points, with a standard normal prior.

```
quad <- seq(-4, 4, length.out = 81)
prior <- dnorm(quad)
eap <- function(a, b, x) {
  like <- sapply(quad, function(t) prod(ifelse(x == 1, plogis(a * (t - b)),
                                               1 - plogis(a * (t - b)))))
  post <- like * prior / sum(like * prior)
  est <- sum(quad * post)
  c(theta = est, se = sqrt(sum((quad - est)^2 * post)))
}
cat_one <- function(true_theta, bank, se_stop = 0.3, max_items = 30, top = 1) {
  used <- integer(0); x <- integer(0); est <- c(theta = 0, se = 1)
  while (length(used) < max_items && est["se"] > se_stop) {
    info <- item_info(bank$a, bank$b, est["theta"]); info[used] <- -Inf
    best <- order(info, decreasing = TRUE)[seq_len(top)]
    item <- if (top == 1) best else sample(best, 1)   # randomesque exposure control
    used <- c(used, item)
    x <- c(x, rbinom(1, 1, plogis(bank$a[item] * (true_theta - bank$b[item]))))
    est <- eap(bank$a[used], bank$b[used], x)
  }
  list(theta = est[["theta"]], se = est[["se"]], n = length(used), items = used)
}
```

Simulate 300 test takers across the ability range. (To run the CAT on your own bank, replace `bank` in the data chunk of Example 1; it needs columns `item`, `area`, `a` and `b`.)

```
# Simulated test takers for the CAT: one true theta per person.
set.seed(1112)
true_theta <- rnorm(300)
```

```
runs <- lapply(true_theta, cat_one, bank = bank)
res <- data.frame(true = true_theta,
                  est = sapply(runs, `[[`, "theta"),
                  se = sapply(runs, `[[`, "se"),
                  n = sapply(runs, `[[`, "n"))
c(mean_length = mean(res$n), rmse = sqrt(mean((res$est - res$true)^2)),
  cor = cor(res$est, res$true))
#> mean_length        rmse         cor
#>      12.337       0.292       0.959
```

On average the CAT reaches a standard error of 0.3 with far fewer than 30 items. Test length depends on where a person is: In the middle, where the bank is rich, the CAT stops quickly; at the extremes it runs out of informative items and goes to the maximum.

```
ggplot(res, aes(true, n)) + geom_point(alpha = 0.5) +
  labs(x = "True theta", y = "Items administered") + theme_mirtverse()
```

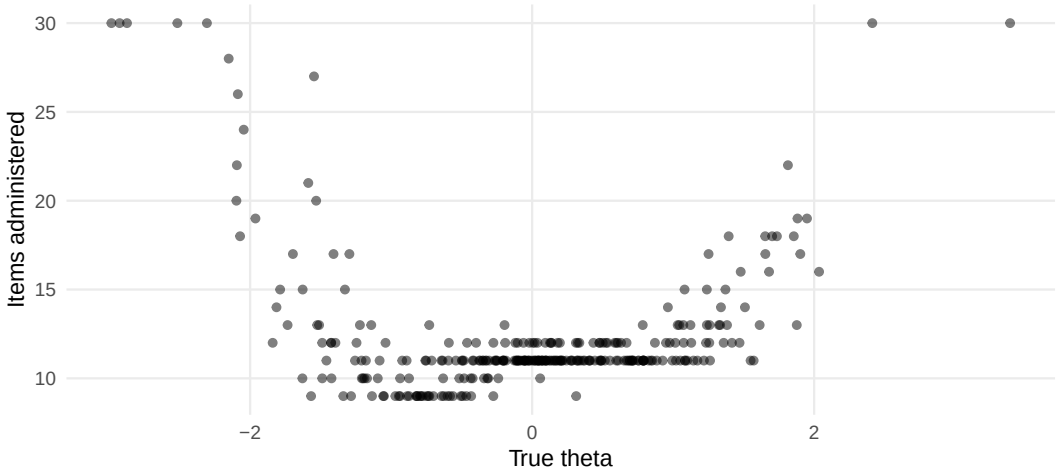


Figure 11.2: Number of items administered by the CAT against true ability.

Exposure is the other side of the coin. With pure maximum-information selection, a few items are given to a large share of test takers:

```
exposure <- function(runs) {
  table(factor(unlist(lapply(runs, `[[`, "items")), levels = 1:200)) / length(runs)
}
exp_max <- exposure(runs)
```

```
sort(round(exp_max, 2), decreasing = TRUE)[1:6]
#>
#>   31  100   97   67  135  140
#> 1.00 0.68 0.64 0.60 0.48 0.48
```

The first item is the same for everybody (the most informative item at $\theta = 0$), so its exposure is 100%. The **randomesque** method picks at random among the five most informative items, which spreads exposure at a small cost in test length:

```
set.seed(1113)
runs_r <- lapply(true_theta, cat_one, bank = bank, top = 5)
exp_r <- exposure(runs_r)
rbind(max_info = c(max_exposure = max(exp_max), mean_length = mean(sapply(runs,
 ↪  `[[`, "n"))),
      randomesque = c(max_exposure = max(exp_r), mean_length = mean(sapply(runs_r,
       ↪  `[[`, "n"))))
#>             max_exposure mean_length
#> max_info           1.000        12.3
#> randomesque        0.677        12.9
```

Content balancing (for example, cycling through the four areas) works the same way: Restrict the candidate items before choosing the most informative one. `simulate_cat()` in the assessment pipeline runs this kind of simulation on a real bank, and the mirtCAT package can deliver an operational CAT.

11.4 Your turn

1. Add 20 hard items ($b \sim N(2.2, 0.4)$) to the bank in Example 1 and redraw the information curve. What happens to CAT length for the most able test takers?
2. Change the stopping rule to $SE < 0.25$. How do mean length and RMSE change?
3. Add content balancing to `cat_one()` so that each test has at least four items from every area. (Hint: While an area has fewer than four items, restrict the candidates to that area.)
4. An admissions officer asks why two applicants got different questions and different numbers of questions. Write a short explanation.

Solutions

```
set.seed(1114)
hard <- data.frame(item = sprintf("H%02d", 1:20), area = factor("Idioms",
 ↪  levels(bank$area)),
                   a = rlnorm(20, 0.2, 0.3), b = rnorm(20, 2.2, 0.4))
bank2 <- rbind(bank, hard)
```

```
top <- true_theta > 1.5
mean(sapply(lapply(true_theta[top], cat_one, bank = bank2), `[[`, "n"))
#> [1] 14.8
mean(res$n[top])
#> [1] 17.8

runs25 <- lapply(true_theta, cat_one, bank = bank, se_stop = 0.25)
c(mean_length = mean(sapply(runs25, `[[`, "n")),
  rmse = sqrt(mean((sapply(runs25, `[[`, "theta") - true_theta)^2)))
#> mean_length        rmse
#>      18.963       0.246
```

1. The CAT reaches the target precision sooner for the most able test takers, because it now has informative items for them.
2. Tests get longer and estimates more precise: Precision costs items.
3. For example: "The test adapts to each applicant: After every answer it picks the next question that tells us most about that applicant. Everyone is measured on the same scale, and the test stops when it has measured each person precisely enough, which takes fewer questions for some people than for others."

In the lab

> Lab 5, `run_lab("cat-design")`, asks whether a college should replace its fixed math placement test with a CAT. You examine the bank, run a CAT by hand, simulate it, and set exposure, content and stopping rules.

Further reading

Wainer and colleagues (2000) give an accessible introduction to CAT; van der Linden (2005) covers optimal test assembly.

Chapter 12

Cut scores, reporting and communicating results

Learning goals. After this chapter you can

- describe the Bookmark standard-setting method and build an item map at a response probability of 0.67;
- turn a cut on the θ scale into a raw-score cut;
- estimate classification accuracy and consistency at a cut score;
- report IRT results to people who do not use R, with a written summary, a report and a dashboard.

Key terms: cut score, standard setting, response probability, classification accuracy, classification consistency, test characteristic curve.

12.1 The ideas

A **cut score** divides a scale into categories such as *pass* and *fail*, or *below basic*, *proficient* and *advanced*. Where the cut goes is a judgement about what performance is good enough, made by people who know the content and the consequences. **Standard setting** is the family of structured procedures for making that judgement defensibly (Cizek and Bunch 2007).

The **Bookmark method** is closely tied to IRT. Items are ordered from easiest to hardest in an *ordered item booklet*, where each item's location is the θ at which a test taker has a fixed probability (usually 0.67, the **response probability** RP67) of answering it correctly. Each panelist places a bookmark after the last item that a *just-qualified* candidate should master. The location of the bookmarked item is that panelist's cut, and the panel's cut is usually the median over panelists after several rounds of discussion. For a 2PL item, the RP67 location is

$$\theta_{0.67} = b + \frac{\ln(0.67/0.33)}{a}.$$

Once a cut is fixed on the θ scale, two numbers describe how well the test sorts people. **Classification accuracy** is the proportion of people whose estimated category matches their true category. **Classification consistency** is the proportion who would be placed in the same category on two independent administrations of the test. Both can be estimated from a single administration using the scores and their standard errors (Rudner 2001).

Finally, results have to be communicated. People who make decisions with test scores rarely read item parameters. They need a short, honest summary of what the test measures, how precisely, for whom it works, and what the scores mean.

12.2 Example 1: Setting a proficiency standard

A provincial Grade 6 science test has 36 items. We simulate a calibration on 2,500 students.

```
# Simulated 36-item science test, 2,500 students.
set.seed(1212)
a <- matrix(rlnorm(36, 0.15, 0.25)); b <- sort(rnorm(36, 0, 1))
resp <- simdata(a, matrix(-a * b), N = 2500, itemtype = "2PL")
colnames(resp) <- sprintf("S%02d", 1:36)
```

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
pars <- coef(mod, IRTpars = TRUE, simplify = TRUE)$items
booklet <- data.frame(item = rownames(pars), a = pars[, "a"], b = pars[, "b"])
booklet$rp67 <- booklet$b + log(0.67 / 0.33) / booklet$a
booklet <- booklet[order(booklet$rp67), ]
booklet$page <- seq_len(nrow(booklet))
head(booklet, 8)
#>     item     a      b   rp67 page
#> S02  S02 1.924 -1.473 -1.105    1
#> S01  S01 0.928 -1.827 -1.064    2
#> S03  S03 1.243 -1.387 -0.818    3
#> S05  S05 1.335 -0.955 -0.425    4
#> S06  S06 1.118 -1.011 -0.377    5
#> S04  S04 0.833 -1.090 -0.239    6
#> S08  S08 1.245 -0.741 -0.173    7
#> S10  S10 1.931 -0.488 -0.121    8
```

The ordered item booklet is `booklet`, one page per item. Suppose eight panelists, after two rounds of discussion, place their bookmarks for *proficient* on these pages:

```
bookmarks <- c(20, 22, 21, 23, 22, 24, 21, 22)
panel_cuts <- booklet$rp67[bookmarks]
theta_cut <- median(panel_cuts)
c(theta_cut = theta_cut, spread = sd(panel_cuts))
#> theta_cut    spread
```

```
#>    0.7943    0.0586
```

The standard deviation of the panelists' cuts, divided by the square root of the number of panelists, is a rough standard error of the panel's cut. It should be reported alongside the cut.

Teachers and parents see raw scores, so we convert the cut with the **test characteristic curve**: The expected number correct at θ.

```
raw_cut <- expected.test(mod, matrix(theta_cut))
c(theta_cut = theta_cut, expected_raw = raw_cut, raw_cut = ceiling(raw_cut))
#>    theta_cut expected_raw      raw_cut
#>        0.794       24.583       25.000
```

How many students are classified as proficient, and how accurately?

```
sc <- fscores(mod, full.scores.SE = TRUE)
above <- sc[, 1] >= theta_cut
p_above <- pnorm((sc[, 1] - theta_cut) / sc[, 2])
c(percent_proficient = mean(above),
  accuracy = mean(ifelse(above, p_above, 1 - p_above)),
  consistency = mean(p_above^2 + (1 - p_above)^2))
#> percent_proficient           accuracy        consistency
#>              0.204              0.925              0.893
```

Accuracy compares each student's category with where their true score probably lies; consistency asks how often two independent testings would agree (here approximated from each student's posterior probability of being above the cut). Consistency is always the lower of the two, because both testings carry error.

An item map makes the cut concrete for teachers: It shows which items a just-proficient student is likely to answer correctly (the items below the line). Printing the text of a few items on each side of the line is a good way to describe what *proficient* means.

```
library(ggplot2)
ggplot(booklet, aes(page, rp67, colour = rp67 <= theta_cut)) +
  geom_hline(yintercept = theta_cut, colour = "#C0392B", linewidth = 0.8) +
  geom_point(size = 2) +
  annotate("text", x = 1, y = theta_cut + 0.15, label = "Proficient cut",
           colour = "#C0392B", hjust = 0, size = 3.5) +
  scale_colour_manual(values = c(`TRUE` = "#2C5F8A", `FALSE` = "grey60"),
                      labels = c(`TRUE` = "mastered at the cut", `FALSE` = "not
                       ↪  yet")) +
  labs(x = "Page in the ordered item booklet", y = "RP67 location (theta)",
       colour = NULL) +
  theme_mirtverse()
```

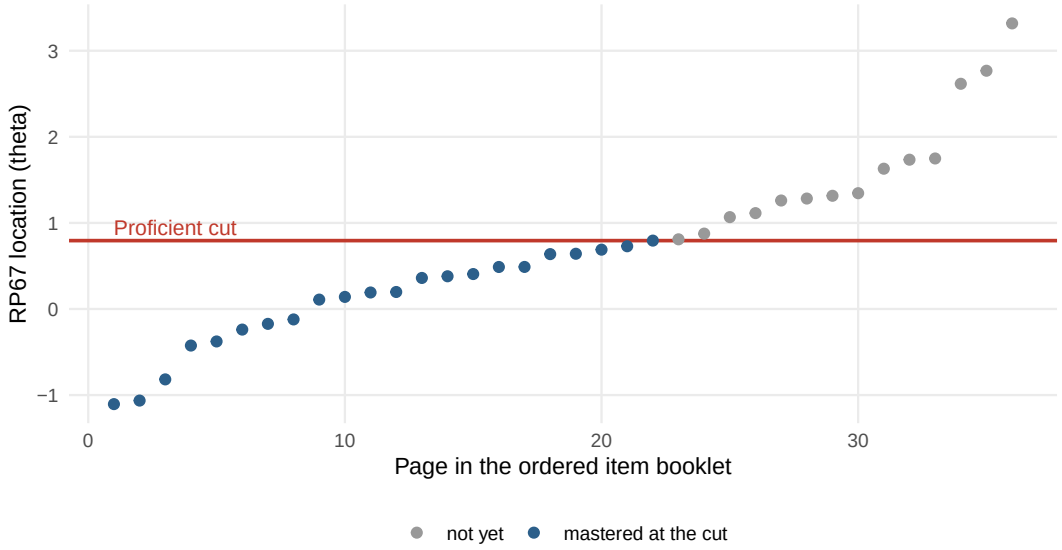


Figure 12.1: Item map of the ordered item booklet with the proficient cut score.

In a real programme, `item_map()` and `set_cuts()` in the assessment pipeline produce the booklet, record the panel's judgements, and report impact and classification statistics for several cuts at once.

12.3 Example 2: Telling the story

We return to the attitude scale from Chapter 8 to show the reporting tools on a different kind of instrument.

```
# Four 4-category attitude items (scored 1-4), 392 respondents.
resp <- Science
```

`select_model()` and `interpret()` give a plain-language summary that is a good first draft for the methods and results sections of a report:

```
sel <- select_model(resp, verbose = FALSE)
interpret(sel$best)
#> Overview
#> --------
#> * A graded model with 1 dimension was fitted to 392 respondents and 4 items
#>   (EM estimation).
#> * The estimation converged normally.
#>
#> Model fit
#> ---------
#> * Overall model fit is poor: C2 = 19.18, df = 2, RMSEA = 0.148, CFI = 0.910,
```

```
#>   SRMSR = 0.073.
#>
#> Reliability
#> -----------
#> * Marginal reliability is 0.67 (low).
#> * Scores are too imprecise for decisions about individuals; consider adding
#>   items or using the scores only for group-level summaries.
#>
#> Targeting
#> ---------
#> * The average item location is -0.99 and the average person score is 0.00, so
#>   the test is relatively easy for this group.
#> * Measurement is most precise around theta = -2.20 (information = 2.42, SE =
#>   0.64).
#>
#> Items
#> -----
#> * No items were flagged for low discrimination, extreme location, or misfit.
#>
#> Local dependence
#> ----------------
#> * 1 item pair shows possible local dependence (Q3 exceeds the average of
#>   -0.19 by more than 0.2).
#> * Strongest: Comfort & Benefit (Q3 = 0.08).
#> * Check whether these items share a stimulus, wording, or content; consider a
#>   testlet or bifactor model.
#>
#> Person fit
#> ----------
#> * 4.8% of respondents have unusual response patterns (Zh < -1.96), which is
#>   within what chance alone would produce.
```

Read it critically before you use it: It flags the poor absolute fit and the low precision, and a careful report says what that means for the intended use of the scores.

`interpret_prompt()` packages the same results into a prompt you can paste into an AI assistant to draft a longer narrative. Nothing is sent anywhere.

For people who want to see everything, mirtverse writes a complete HTML report in one call, with the model comparison, the interpretation, tables, figures, a sum-score conversion table, item health cards and the code to reproduce it:

```
mv_report(sel, file = "science-attitudes-report.html")
```

For people who want to explore, a dashboard lets them move an ability slider, click through

items and download tables, without R:

```
mv_dashboard(sel$best, title = "Attitudes to science")
export_dashboard(sel$best, "science-dashboard.html", format = "html")
```

See Sharing results with a dashboard for the options.

12.3.1 A one-page memo

Decision makers usually want one page. A structure that works:

1. **The question.** What decision will the scores support?
2. **The answer.** One or two sentences: Can the test support it?
3. **The evidence.** Three to five bullets: Precision where the decision is made, classification accuracy, fairness checks, any items removed.
4. **What it means.** The consequences of the cut for the people tested, for example the percentage classified in each category and how many are close to the cut.
5. **Recommendations and limits.**

12.4 Your turn

1. In Example 1, how many raw-score points does the cut move if every panelist moves their bookmark one page later?
2. Repeat the item map with RP50 instead of RP67. Does the cut move up or down? Why does the choice of response probability matter to panelists?
3. Add a second cut for *advanced* at page 30. Estimate the accuracy of the three-category classification.
4. Write the "answer" and "evidence" parts of a memo about the proficiency cut in Example 1.

Solutions

```
cut_later <- median(booklet$rp67[bookmarks + 1])
c(expected.test(mod, matrix(theta_cut)), expected.test(mod, matrix(cut_later)))
#> [1] 24.6 24.7

rp50 <- sort(booklet$b)
c(rp67_cut = theta_cut, rp50_cut = median(rp50[bookmarks]))
#> rp67_cut rp50_cut
#>    0.794    0.100

adv <- median(booklet$rp67[rep(30, 8)])
cats <- cut(sc[, 1], c(-Inf, theta_cut, adv, Inf), labels = c("below",
 ↪  "proficient", "advanced"))
p_cat <- cbind(pnorm((theta_cut - sc[, 1]) / sc[, 2]),
```

```
              pnorm((adv - sc[, 1]) / sc[, 2]) - pnorm((theta_cut - sc[, 1]) / sc[,
                ↪  2]),
              1 - pnorm((adv - sc[, 1]) / sc[, 2]))
mean(p_cat[cbind(seq_len(nrow(p_cat)), as.integer(cats))])
#> [1] 0.88
```

1. Hardly at all here, because the items in that part of the booklet have very similar RP67 locations. Where the booklet has gaps between items, a single page can move the cut a lot, which is worth showing panelists.
2. With RP50 every item's location is lower, so the same bookmark gives a lower cut. Panelists must understand what "mastered" means (a 50% or a 67% chance), or their bookmarks will not mean what they intend.
3. For example: "The test can support a proficient/not proficient decision. About 93% of students are expected to be classified correctly, and 89% would be classified the same way if tested again. The panel's eight judgements were close together, and about one student in five is classified as proficient." Use the numbers from your own output, and add the standard error at the cut (`1 / sqrt(testinfo(mod, matrix(theta_cut)))`).

In the lab

> The course map ends with a final project: A full analysis of a dataset of the student's choice, handed in as an `mv_report()` or a dashboard plus a two-page memo. The assessment pipeline is a useful checklist for it.

Further reading

Cizek and Bunch (2007) describe the Bookmark method and other standard-setting procedures. Rudner (2001) shows how to estimate classification accuracy from IRT results. The *Standards* (American Educational Research Association et al. 2014) set out what to report about cut scores.

Part I

AI in measurement

Chapter 13

Machine learning through an IRT lens

Learning goals. After this chapter you can

- explain the difference between a **prediction model** and a **measurement model**, and say what each can and cannot claim;
- show that a 2PL item response function is a logistic regression on a latent predictor, and what goes wrong when an observed score stands in for it;
- estimate how well a model predicts new cases with **cross-validation**, and recognize **overfitting**;
- explain why a single IRT score often predicts better than all the item responses fed to a flexible model.

Key terms: machine learning, feature, overfitting and cross-validation, latent trait. The translation table pairs many more machine learning words with their psychometric relatives.

13.1 The ideas

Machine learning (ML) is a set of methods for learning a rule that predicts an outcome from data. It is judged by one question: How well does the rule predict cases it has not seen? Breiman (2001) called this the *algorithmic* culture of statistics, in contrast to the *data modelling* culture that psychometrics grew up in, where a model is a claim about how the data arose and its parameters mean something.

An IRT model makes such a claim. The 2PL says that the chance of a correct answer depends on one unobserved trait through a logistic curve whose slope and location are properties of the item. That is a logistic regression, with one twist: The predictor θ is never observed. Everything that makes IRT useful follows from taking that claim seriously: Item parameters that do not depend on who took the test, a standard error for every score, information to design tests with, and checks of fit that can reject the model.

A prediction model makes a weaker promise. It does not say why the outcome happens, only that its rule will predict well in data like the training data. In exchange it can use any predictors

(**features**) in any flexible form. Two habits of ML are worth taking home even if you never fit a neural network:

- **Hold data out.** A model always fits the data it was estimated on better than new data, and a flexible model can fit noise. This is **overfitting**, and the cure is to judge a model on data it did not see. **Cross-validation** splits the data into k parts (folds), fits the model on $k - 1$ of them, predicts the one left out, and repeats until every case has been predicted once (James et al. 2021).
- **Shrink.** Penalizing large coefficients (ridge, lasso) trades a little bias for a large cut in variance. It is the same idea as a prior on item parameters, or as MAP and EAP scoring pulling extreme scores toward the mean.

Yarkoni and Westfall (2017) make the case that psychology should care more about prediction. The examples below show the reverse lesson as well: A good measurement model is itself a very effective way to predict.

13.2 Example 1: The 2PL is a logistic regression on theta

We simulate 20 items answered by 2,000 test takers whose true abilities we keep, so we can do something impossible with real data: Regress each item on the true θ.

```
# Simulated 20-item test with known abilities and item parameters.
set.seed(1313)
n <- 2000
theta <- rnorm(n)
a <- round(rlnorm(20, log(1.3), 0.3), 2)
b <- round(sort(rnorm(20, 0, 1)), 2)
p <- plogis(outer(theta, a) - matrix(a * b, n, 20, byrow = TRUE))
resp <- as.data.frame((matrix(runif(n * 20), n) < p) * 1L)
names(resp) <- sprintf("Q%02d", 1:20)
```

A logistic regression of item 5 on the true θ estimates its slope and intercept directly; on the logit scale the intercept is $-ab$:

```
fit5 <- glm(resp$Q05 ~ theta, family = binomial)
c(slope = unname(coef(fit5)[2]), b = unname(-coef(fit5)[1] / coef(fit5)[2]),
  true_a = a[5], true_b = b[5])
#>  slope      b true_a true_b
#>  1.328 -0.139  1.210 -0.190
```

With real data we do not have θ. The obvious stand-in is the sum score. Doing the same regression on the standardized rest score (the sum of the other 19 items) gives slopes that are on average about a fifth too flat, because the rest score measures θ with error (Figure 13.1). IRT avoids this by estimating θ and the item parameters together:

```
mod <- mirt(resp, 1, itemtype = "2PL", verbose = FALSE)
est <- coef(mod, IRTpars = TRUE, simplify = TRUE)$items
slope_obs <- sapply(1:20, function(j) {
  rest <- scale(rowSums(resp[, -j]))
  coef(glm(resp[[j]] ~ rest, family = binomial))[2]
})
round(c(true = mean(a), irt = mean(est[, "a"]), rest_score = mean(slope_obs)), 2)
#>       true        irt rest_score
#>       1.28       1.26       1.05
```

```
slopes <- rbind(data.frame(true = a, estimate = est[, "a"], method = "2PL (mirt)"),
                data.frame(true = a, estimate = slope_obs, method = "Regression on
                 ↪  rest score"))
ggplot2::ggplot(slopes, ggplot2::aes(true, estimate, colour = method)) +
  ggplot2::geom_abline(linetype = 2, colour = "grey50") +
  ggplot2::geom_point(size = 2) +
  ggplot2::labs(x = "True slope (a)", y = "Estimated slope", colour = NULL) +
  theme_mirtverse()
```

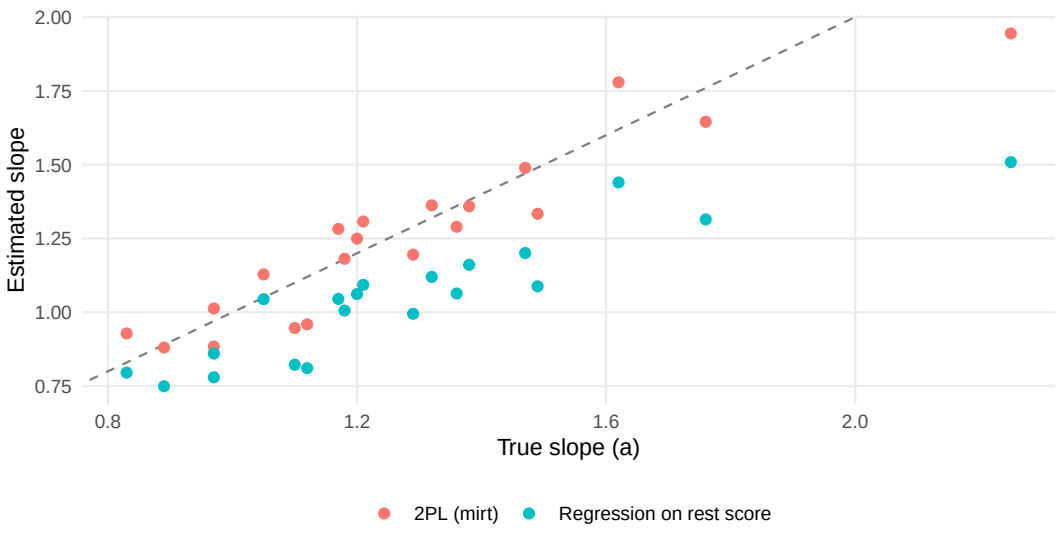


Figure 13.1: Item slopes estimated by the 2PL and by logistic regression on the standardized rest score, against the true slopes. The dashed line marks perfect recovery.

The lesson carries over to every ML model that uses test scores as features: An observed score is a noisy measure of the trait, and models that ignore the noise underestimate how strongly the trait relates to anything else.

13.3 Example 2: Forty items or one score?

A university wants to predict which applicants will pass a first-year statistics course from a 40-item placement test. We simulate 400 past students whose course result depends only on their ability, plus noise.

```
# Simulated 40-item placement test and a pass/fail course outcome.
set.seed(2027)
n <- 400
ability <- rnorm(n)
a2 <- rlnorm(40, log(1.1), 0.3)
b2 <- rnorm(40, 0, 1)
p2 <- plogis(outer(ability, a2) - matrix(a2 * b2, n, 40, byrow = TRUE))
resp <- as.data.frame((matrix(runif(n * 40), n) < p2) * 1L)
names(resp) <- sprintf("P%02d", 1:40)
passed <- rbinom(n, 1, plogis(1.6 * ability + 0.3))
```

The ML approach feeds all 40 item responses to a classifier as features. The measurement approach first summarizes them in one IRT score and uses only that. Both are logistic regressions here, so the comparison is about the features, not the algorithm. We judge each by its accuracy on the data it was fitted to and with 10-fold cross-validation:

```
theta_hat <- fscores(mirt(resp, 1, verbose = FALSE))[, 1]
dat <- data.frame(passed, theta_hat, resp)
accuracy <- function(fit, newdata)
  mean((predict(fit, newdata, type = "response") > 0.5) == newdata$passed)

cv <- function(formula, k = 10) {
  fold <- sample(rep(1:k, length.out = nrow(dat)))
  mean(sapply(1:k, function(f) {
    fit <- suppressWarnings(glm(formula, binomial, data = dat[fold != f, ]))
    accuracy(fit, dat[fold == f, ])
  }))
}
items_formula <- reformulate(names(resp), "passed")
set.seed(1)
data.frame(
  features = c("40 item responses", "one IRT score"),
  training_accuracy = c(accuracy(suppressWarnings(glm(items_formula, binomial, data
   ↪  = dat)), dat),
                        accuracy(glm(passed ~ theta_hat, binomial, data = dat), dat)),
  cv_accuracy = c(cv(items_formula), cv(passed ~ theta_hat)))
#>            features training_accuracy cv_accuracy
#> 1 40 item responses             0.787       0.720
#> 2     one IRT score             0.735       0.742
```

On the training data the 40-item model looks better: With 41 coefficients it can fit some of the noise. Cross-validation tells the truth. On new students it does worse than the single score, because the IRT model has already done what a good ML pipeline would have had to learn: That the items measure one thing and should be combined with weights close to their discriminations. A measurement model is a strong, theory-based form of regularization.

This is not an argument against ML. With thousands of students, features that carry information beyond the trait (response times, the text of answers), or an outcome that depends on more than one trait, a flexible model may well win. The point is to compare on held-out data and to know what each approach is claiming.

13.4 Your turn

1. In Example 1, repeat the rest-score regression using the EAP score from `fscores(mod)` instead of the rest score. Are the slopes closer to the truth? Why not?
2. In Example 2, change `n` to 2,000 in the data chunk and run the comparison again. What happens to the gap between the two approaches, and why?
3. Shrinking the 40 item coefficients toward a common value reduces overfitting; giving every item the same weight, the proportion correct `rowMeans(resp)`, is the extreme case. How does a model on the proportion correct do under cross-validation in Example 2?
4. A vendor reports that their ML model "predicts course success with 82% accuracy" from your placement test items. Write three questions you would ask before believing it.

Solutions

```
# 1. the same regression on the EAP score of Example 1
resp1 <- extract.mirt(mod, "data")
eap <- fscores(mod)[, 1]
slope_eap <- sapply(1:20, function(j) coef(glm(resp1[, j] ~ eap, family =
 ↪  binomial))[2])
round(c(true = mean(a), eap = mean(slope_eap)), 2)
#> true  eap
#> 1.28 1.56

# 3. equal weights: the extreme form of shrinkage
dat$prop <- rowMeans(resp)
set.seed(1)
cv(passed ~ prop)
#> [1] 0.74
```

1. Now the slopes are too *steep* on average. EAP scores are shrunk toward the mean, so their spread is smaller than θ's and each unit of EAP corresponds to more than one unit

of θ; and each item's own response helped compute the EAP score it is regressed on. An estimated score is not θ, whichever way you compute it; that is why IRT estimates the item parameters and the trait distribution together.

2. With 2,000 students the 40-item model has more data per coefficient, so it overfits less and the cross-validated gap narrows. It still rarely beats the IRT score, because the outcome depends only on the trait.
3. The proportion correct does about as well as the IRT score under cross-validation: Equal weights are a strong form of shrinkage and lose little when the items' discriminations are similar.
4. For example: Was the 82% computed on students who were not used to train the model? What is the accuracy of predicting from the total score alone, or of predicting that everyone passes? Is the accuracy similar for subgroups of students, such as English language learners?

In the AI Hub

> The AI Hub has a learning path through AI in measurement, and its translation table lists the ML terms used in this chapter with their psychometric relatives.

Further reading

James et al. (2021) is the standard introduction to statistical learning, with R labs. Breiman (2001) and Yarkoni and Westfall (2017) are short, readable arguments for taking prediction seriously. Bulut et al. (2024) survey where AI is used in educational measurement.

Chapter 14

What item text tells us

Learning goals. After this chapter you can

- explain why predicting item difficulty from text is useful, and what it can and cannot replace;
- describe items by **text features** with `item_features()`, and connect them to the item covariates of the **linear logistic test model** (LLTM);
- predict difficulty with an explanatory model, a lasso and a random forest using `predict_difficulty()`, and compare them by **cross-validation**;
- fit the one-step LLTM to responses in mirt;
- recognize **leakage**, choose the right unit to hold out, and know the limit set by the error in estimated difficulties.

Key terms: Feature, LLTM, overfitting and cross-validation, embedding, data leakage.

14.1 The ideas

Every new item needs a difficulty before it can be used for scoring, and the usual way to get one is a **field test**: The item is given to hundreds of test takers and calibrated (Chapter 3). Field testing is slow and expensive, and it exposes items before they count. Item writers also know that some features make items harder: More reading, an extra step, a negation, decimals instead of whole numbers. If difficulty could be **predicted from the text**, a program could target field-test forms, use smaller field-test samples, steer item writing towards the difficulties the bank lacks, and judge items written by a generator (Chapter 17).

Psychometrics and machine learning both have a way of doing this.

- **Explanatory IRT.** The linear logistic test model (Fischer 1973) replaces each item's difficulty with a weighted sum of its features:

$$b_i = \eta_0 + \sum_k \eta_k q_{ik},$$

where q_{ik} is the value of feature k for item i (the item's covariates) and η_k is how much the feature adds to difficulty, in logits. The weights are read as evidence about *what makes items hard*, which is a claim about the construct (De Boeck and Wilson 2004). The model can be fitted to the responses in one step, or in two steps by regressing estimated difficulties on the features.

- **Prediction.** Machine learning treats difficulty as a label to predict from as many features as are useful: Counts and readability measures, word frequencies, or **embeddings** of the text from a language model. Models such as the lasso or a random forest are judged by how well they predict items they were not trained on (AlKhuzaey et al. 2024; Benedetto et al. 2023).

The two are less different than they look. The two-step LLTM *is* a linear regression, and the lasso is the same regression with its weights shrunk. What differs is the goal: Explaining difficulty, or predicting it as accurately as possible. Three cautions apply to both.

- **Leakage.** A feature computed from the same responses as the difficulty, such as the proportion correct in the field test, predicts difficulty almost perfectly and is useless, because it is not available until the item has been field-tested.
- **What is held out.** Items written from the same template (an item model) share most of their difficulty. A model that has seen other items from an item's template predicts it well; a model asked about a new kind of item usually does much worse. Cross-validation must hold out the unit you will predict for.
- **The noise floor.** Estimated difficulties have standard errors, and features never capture everything that makes an item hard. No model can predict b more closely than its estimation error, and an LLTM rarely fits as well as giving every item its own difficulty. Real benchmarks show modest gains over simple baselines (Yaneva et al. 2024).

14.2 Example 1: Features and an explanatory model

A science program has 70 multiple-choice items of the form *"Which of these is a mammal?"*, on seven topics. Half the items are negated (*"Which of these is NOT a mammal?"*), half start with a context sentence that adds reading but no content, and in some the key happens to be the longest option, a cue that test-wise students use. We simulate the items and the responses of 1,000 students, and calibrate the Rasch model.

```
# Simulated science classification items, with responses of 1,000 students.
set.seed(1414)
groups <- list(   # members, non-members and base difficulty of each topic
  mammal = list(yes = c("whale", "bat", "dolphin", "rabbit", "horse", "squirrel"),
                no = c("shark", "penguin", "frog", "crocodile", "salmon", "eagle"),
                 ↪  b = -0.8),
  reptile = list(yes = c("lizard", "turtle", "snake", "crocodile", "gecko",
   ↪  "tortoise"),
                 no = c("frog", "salamander", "newt", "toad", "eel", "seal"), b =
                  ↪  0.2),
```

```
  insect = list(yes = c("ant", "beetle", "butterfly", "grasshopper", "bee",
  ↪  "dragonfly"),
               no = c("spider", "scorpion", "centipede", "tick", "crab", "snail"),
                ↪  b = 0),
  metal = list(yes = c("copper", "iron", "aluminium", "zinc", "silver", "tin"),
              no = c("glass", "wood", "plastic", "rubber", "sulfur", "carbon"), b =
               ↪  -0.3),
  conductor = list(yes = c("copper wire", "a steel nail", "an aluminium can", "salt
  ↪  water",
                           "graphite", "a silver spoon"),
                  no = c("a rubber band", "dry wood", "a glass rod", "a plastic
                   ↪  ruler", "air",
                         "pure water"), b = 0.5),
  renewable = list(yes = c("wind", "sunlight", "tides", "geothermal heat", "moving
  ↪  water", "biomass"),
                  no = c("coal", "oil", "natural gas", "uranium", "peat",
                   ↪  "diesel"), b = -0.4),
  physical = list(yes = c("ice melting", "sugar dissolving in water", "glass
  ↪  breaking",
                          "water boiling", "wax melting", "salt crystals forming as
                           ↪  water evaporates"),
                 no = c("wood burning", "iron rusting", "milk souring", "bread
                  ↪  baking",
                        "a match burning", "baking soda fizzing in vinegar"), b =
                        ↪  0.9))
label <- c(mammal = "a mammal", reptile = "a reptile", insect = "an insect", metal
↪  = "a metal",
           conductor = "a good conductor of electricity", renewable = "a renewable
            ↪  energy source",
           physical = "an example of a physical change")
contexts <- c("A class is sorting cards for a science display.",
              "Ms Okafor gives her students a quick review question.",
              "Students are making a poster for the school science fair.")
rows <- list()
for (g in names(groups)) for (i in 1:10) {
  neg <- i > 5
  gr <- groups[[g]]
  key <- sample(if (neg) gr$no else gr$yes, 1)
  opts <- sample(c(key, sample(if (neg) gr$yes else gr$no, 3)))
  stem <- sprintf("Which of these is %s%s?", if (neg) "NOT " else "", label[[g]])
  if (i %% 2 == 0) stem <- paste(sample(contexts, 1), stem)
  rows[[length(rows) + 1]] <- data.frame(
    id = sprintf("S%s%02d", toupper(substr(g, 1, 3)), i), topic = g, stem = stem,
```

```
    A = opts[1], B = opts[2], C = opts[3], D = opts[4], key = LETTERS[match(key,
     ↪  opts)])
}
sci <- do.call(rbind, rows)
f <- item_features(sci)
b_true <- sapply(sci$topic, function(g) groups[[g]]$b) + 0.7 * f$negation +
  0.25 * (f$sentences - 1) - 0.3 * f$key_longest + rnorm(nrow(sci), 0, 0.25)
theta <- rnorm(1000)
resp <- sapply(b_true, function(b) rbinom(1000, 1, plogis(theta - b)))
colnames(resp) <- sci$id
rasch <- mirt(as.data.frame(resp), 1, "Rasch", verbose = FALSE)
sci$b <- -coef(rasch, simplify = TRUE)$items[, "d"]   # Rasch difficulty
```

`item_features()` reads the stem and the options and returns one row of features per item: Length, sentences, readability (`grade`), numbers, negation, the length of the options, whether the key is the longest option, and each item's most similar other item.

```
feats <- item_features(sci)
feats[c(1, 2, 6), c("id", "words", "sentences", "grade", "negation",
 ↪  "key_longest")]
#> # A tibble: 3 x 6
#>   id     words sentences grade negation key_longest
#>   <chr>  <int>     <int> <dbl>    <int>       <int>
#> 1 SMAM01     6         1   0.5        0           0
#> 2 SMAM02    15         2   1.5        0           0
#> 3 SMAM06    16         2   4.5        1           1
items <- merge(sci, feats)
round(cor(items[, c("words", "sentences", "grade", "negation", "key_longest")],
 ↪  items$b), 2)
#>              [,1]
#> words        0.44
#> sentences    0.24
#> grade        0.33
#> negation     0.58
#> key_longest -0.16
```

Negated items are harder, and so are longer ones; a longest key goes with easier items. Now predict difficulty from four text features with three models. `predict_difficulty()` fits each model five times, each time predicting the fifth of the items it did not see:

```
text <- c("sentences", "negation", "option_words", "key_longest")
m1 <- predict_difficulty(items, text)
m1
#> <mv_difficulty> 70 items, 4 features, 5-fold cross-validation (random folds)
```

```
#>         method RMSE  MAE    r   R2
#>           LLTM 0.42 0.35 0.76 0.57
#>          Lasso 0.42 0.35 0.76 0.57
#>  Random forest 0.50 0.40 0.73 0.41
#> Use tidy() for the coefficients, autoplot() for the predictions.
```

On items they have not seen, the two linear models explain a little under 60% of the variance in difficulty (R^2), with a typical error (RMSE) of about 0.4 logits. The random forest does worse: With 70 items and effects that simply add up, its flexibility buys nothing. The LLTM's weights say what the features do:

```
tidy(m1)[tidy(m1)$method == "lltm", ]
#> # A tibble: 5 x 4
#>   method term         estimate std_error
#>   <chr>  <chr>           <dbl>     <dbl>
#> 1 lltm   (Intercept)    -0.835    0.171
#> 2 lltm   sentences       0.181    0.0968
#> 3 lltm   negation        0.670    0.100
#> 4 lltm   option_words    0.478    0.0619
#> 5 lltm   key_longest    -0.292    0.128
```

A negation adds about two thirds of a logit, a context sentence about 0.2, and a longest key makes an item easier, as item writers would expect. `option_words` has a large weight too, which is less plausible as a cause: Long options belong to the harder topics, such as physical changes. The features know nothing about topic, so they stand in for it where they can. Adding the topic as a feature brings the error down to about 0.29 logits (Exercise 1), close to the 0.25 of item-specific variation built into the simulation, and the weight of `option_words` shrinks to almost nothing.

14.3 Example 2: The one-step LLTM in mirt

The two-step LLTM treats the estimated difficulties as known. The one-step LLTM fits the features to the responses directly: Each item's difficulty is constrained to the weighted sum of its features. In mirt, the features are passed as `itemdesign` and the model for them as `item.formula`. Here each topic gets its own intercept:

```
design <- data.frame(topic = factor(sci$topic), feats[, c("sentences", "negation",
 ↪  "key_longest")])
lltm <- mirt(as.data.frame(resp), 1, "Rasch", itemdesign = design,
             item.formula = ~ 0 + topic + sentences + negation + key_longest,
              ↪  verbose = FALSE)
v <- mod2values(lltm)
eta <- v[v$item == sci$id[1] & v$name %in% c("sentences", "negation",
 ↪  "key_longest"),
```

```
        c("name", "value")]
eta$difficulty <- -eta$value   # mirt models easiness; reverse the sign for
 ↪  difficulty
eta
#>           name value difficulty
#> 8    sentences -0.21       0.21
#> 9     negation -0.62       0.62
#> 10 key_longest  0.32      -0.32
```

A context sentence adds about 0.2 logits, a negation about 0.6, and a longest key makes an item about 0.3 logits easier: Close to the two-step weights and to the values used to simulate the data. The classic test of an LLTM compares it with the Rasch model, which gives every item its own difficulty:

```
anova(lltm, rasch)
#>         AIC SABIC    HQ   BIC logLik      X2 df p
#> lltm  81315 81334 81336 81369 -40647
#> rasch 80664 80787 80797 81013 -40261 770.604 60 0
```

Ten parameters cannot replace 70 free difficulties, and the likelihood ratio test says so loudly; with 1,000 test takers it nearly always does. The LLTM is still useful: It shows which features matter and by how much. Explanatory IRT models that add a random item effect, so that each item may differ from its features' prediction, combine the two views (De Boeck and Wilson 2004). They are generalized linear mixed models, and the eirm package (Bulut et al. 2021) fits them with lme4 from data in long format, one row per person and item. The model below keeps the LLTM's features and adds a random effect for items, whose variance is the difficulty the features leave unexplained (eirm reports easiness, so the signs are reversed again):

```
# install.packages("eirm")
library(eirm)
long <- data.frame(person = rep(seq_len(nrow(resp)), ncol(resp)),
                   item = rep(colnames(resp), each = nrow(resp)),
                   resp = as.vector(resp))
long <- merge(long, data.frame(item = sci$id, design))
m_eirm <- eirm("resp ~ -1 + topic + sentences + negation + key_longest + (1|person)
 ↪  + (1|item)",
               data = long)
print(m_eirm, difficulty = TRUE)   # fixed effects as difficulties
lme4::VarCorr(m_eirm$model)       # person and residual item variance
```

The same framework adds person covariates, such as language status, and their interactions with item features, which is DIF explained by features rather than item by item.

14.4 Example 3: New items or new kinds of items?

So far each fold held out random items, so every topic was in training. A new item on a known topic is one question; a topic the model has never seen is another. Hold out whole topics with `group`:

```
m3 <- predict_difficulty(items, c("topic", "sentences", "negation", "key_longest"),
                         group = "topic", methods = "lltm")
m3$accuracy
#> # A tibble: 1 x 5
#>   method  rmse   mae     r    r2
#>   <chr>  <dbl> <dbl> <dbl> <dbl>
#> 1 lltm   0.592 0.473 0.445 0.153
```

With its topic held out, an item is predicted as if it belonged to an average topic, and the error doubles to about 0.6 logits; R^2 falls to about 0.15. The features still capture negation, the context sentence and the longest key, but the topic, the largest part of an item's difficulty, cannot be read off the text. This is the usual finding: Text features place items within a family much better than they predict how hard a new kind of item will be. That is why automatic item generation calibrates item models rather than single items. Figure 14.1 shows the predictions of Example 1, with random folds.

```
autoplot(m1)
```

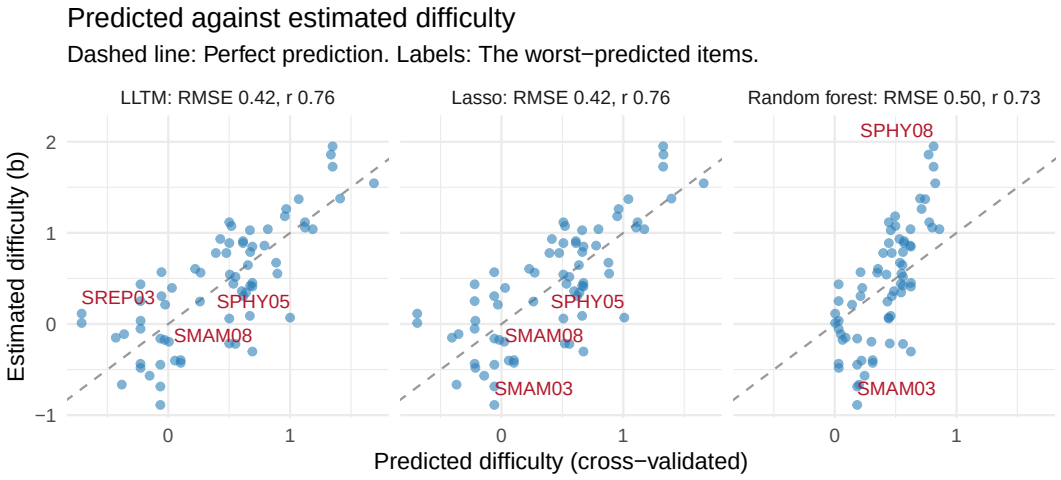


Figure 14.1: Cross-validated predicted against estimated difficulty for the three models of Example 1. Points on the dashed line are predicted exactly; labels mark the worst-predicted items.

14.5 Your turn

1. Add `topic` to the features of Example 1, with random folds. How much does the error drop, and how do the LLTM weights of `negation`, `key_longest` and `option_words` change?
2. A colleague adds `p_pilot`, the proportion correct in this year's field test, to the features. Compute it from `resp`, add it to `items`, and run `predict_difficulty()`. What does the output warn you about, and why is the model useless for new items?
3. Without `topic`, the two-step LLTM with `sentences`, `negation` and `key_longest` gives `key_longest` a weight near zero. Why? Look at `table(items$key_longest, items$topic)`.
4. Look at the `similarity` and `nearest` columns of `feats`. Which items are most alike, and are any of them near-duplicates?

Solutions

```
# 1. topic as a feature
m_topic <- predict_difficulty(items, c("topic", text))
m_topic$accuracy
#> # A tibble: 3 x 5
#>   method  rmse   mae     r    r2
#>   <chr>  <dbl> <dbl> <dbl> <dbl>
#> 1 lltm   0.291 0.240 0.893 0.796
#> 2 lasso  0.288 0.235 0.895 0.800
#> 3 forest 0.410 0.343 0.842 0.593
w <- tidy(m_topic)
w[w$method == "lltm" & !grepl("topic", w$term), ]
#> # A tibble: 5 x 4
#>   method term         estimate std_error
#>   <chr>  <chr>           <dbl>     <dbl>
#> 1 lltm   (Intercept)    -0.135    0.179
#> 2 lltm   sentences       0.214    0.0664
#> 3 lltm   negation        0.633    0.0689
#> 4 lltm   option_words    0.120    0.142
#> 5 lltm   key_longest    -0.337    0.0996

# 2. a leaky feature
items$p_pilot <- colMeans(resp)[items$id]
predict_difficulty(items, c(text, "p_pilot"), methods = "lltm")
#> <mv_difficulty> 70 items, 5 features, 5-fold cross-validation (random folds)
#>  method RMSE  MAE r   R2
#>    LLTM 0.05 0.03 1 0.99
#> Check: `p_pilot` alone has a rank correlation of -1.00 with b. Is it known
 ↪  before field testing, or computed from the same responses?
```

```
#> Use tidy() for the coefficients, autoplot() for the predictions.

# 3. key_longest without topic
tidy(predict_difficulty(items, c("sentences", "negation", "key_longest"), methods =
 ↪  "lltm"))
#> # A tibble: 4 x 4
#>   method term        estimate std_error
#>   <chr>  <chr>          <dbl>     <dbl>
#> 1 lltm   (Intercept)  -0.173      0.204
#> 2 lltm   sentences     0.172      0.133
#> 3 lltm   negation      0.704      0.137
#> 4 lltm   key_longest  -0.0329     0.170
table(items$key_longest, items$topic)
#>
#>     conductor insect mammal metal physical renewable reptile
#>   0        10      9      9     9        5         8       6
#>   1         0      1      1     1        5         2       4

# 4. the most similar items
head(feats[order(-feats$similarity), c("id", "nearest", "similarity")])
#> # A tibble: 6 x 3
#>   id     nearest similarity
#>   <chr>  <chr>        <dbl>
#> 1 SCON06 SCON10       0.759
#> 2 SCON10 SCON06       0.759
#> 3 SREN07 SREN09       0.733
#> 4 SREN09 SREN07       0.733
#> 5 SMET06 SMET10       0.727
#> 6 SMET10 SMET06       0.727
```

1. The RMSE drops from about 0.42 to 0.29 and R^2 rises to about 0.8 for the linear models; the forest stays behind (about 0.41). `negation` (about 0.63) and `key_longest` (about -0.34) are close to the simulated 0.7 and -0.3, while `option_words` falls from about 0.5 to 0.1 with a large standard error: It was standing in for the topic.
2. The model predicts almost perfectly, and the printout flags `p_pilot`: Its rank correlation with b is -1. The proportion correct and the Rasch difficulty are computed from the same responses, so the feature *is* the label in disguise. A new item has no field-test data, so the model cannot be used for it.
3. Five of the 16 items with a longest key are on physical changes, the hardest topic, whose options are long. Without the topic in the model, the easiness that a longest key brings is cancelled by the difficulty of the topic it comes with. An LLTM weight is only as good as the features left out of the model, as in any regression.
4. The most alike pairs (similarity about 0.75) are items on the same topic with the same

stem and context sentence and one or two shared options. They are different items written from one template. A similarity close to 1, with the same options, would mark a near-duplicate that should not appear on the same form.

In the lab

> Lab 8, `run_lab("item-text")`, uses `item_features()` and `predict_difficulty()` on 192 numeracy items, finds what the data hide, and asks how far predicted difficulties can replace a field test. The AI Hub collects the other AI tools in mirtverse.

Further reading

Fischer (1973) introduced the LLTM, and De Boeck and Wilson (2004) place it in the family of explanatory item response models. AlKhuzaey et al. (2024) and Benedetto et al. (2023) review difficulty prediction from text, from the education and the computer science sides. Yaneva et al. (2024) report a shared task on real licensing-exam items, a useful reality check. Bulut, Gorgun and Yildirim-Erbasli (2021) show how to fit explanatory extensions of dichotomous and polytomous Rasch models in R with eirm.

Chapter 15

Generating items

Learning goals. After this chapter you can

- explain **automatic item generation** (AIG) from **item models**, and how it differs from asking a language model for items;
- write an item model with `item_model()`, with distractors built from misconceptions and constraints that rule out flawed items, and generate **isomorphs** with `generate_isomorphs()`;
- check after field testing whether a family of isomorphs is equally difficult with `calibrate_families()`, and find the variable that makes a family spread;
- fit a one-step model with a random item effect in mirt, or with eirm, and say when a family mean can stand in for a field test.

Key terms: Item model, LLTM, calibration.

15.1 The ideas

Testing programs need many items: For new forms each year, for adaptive tests that must not expose the same items too often (Chapter 11), and for practice. Writing and field-testing each item by hand is the slowest and most expensive part of test development. **Automatic item generation** (AIG) writes items from a template (Gierl and Haladyna 2013; Gierl and Lai 2013). An **item model** is a parent item with variable parts: A stem, a key and distractors, each written as a template, the values each variable can take, and constraints that every generated item must meet. Filling in the values gives **isomorphs**: Items that test the same thing in the same way and differ only in their surface, such as the numbers.

Two promises come with item models. The first is quality at scale: If the model is right, every item it generates is right, and an error in the model shows up in every item, where an automatic check can catch it. The second is **calibration at the level of the family**. If isomorphs are equally difficult, the item model can be calibrated once, and its new items can use the family's difficulty without a field test of their own (Glas and van der Linden 2003; Sinharay et al. 2003). That

promise is an empirical claim, and this chapter is about checking it.

A family's items are never exactly equally difficult. The model for a family f is

$$b_i = \mu_f + \varepsilon_i, \qquad \varepsilon_i \sim N(0, \sigma_f^2),$$

where μ_f is the family's mean difficulty and σ_f, the **within-family SD**, says how much its isomorphs differ. A new item's difficulty is predicted by μ_f with an error of about σ_f. When σ_f is small, the family mean is as good as a field test of a few hundred test takers; when it is large, some variable in the item model changes what the items demand, and the model must be split or the variable added as a feature, as in the LLTM of Chapter 14. Explanatory IRT treats this as one model with item features and a random item effect (De Boeck and Wilson 2004; Geerlings et al. 2011).

Language models offer a second route: Ask for items in a prompt (`generate_items()`, Chapter 17). They write more varied items, but each is a one-off, with no template to calibrate and no guarantee that two items with the same prompt are isomorphs. Template-based AIG is predictable and traceable; every item can be rebuilt from its values. In practice the two meet: Language models can help write item models, and their items can be reviewed with the same checks.

15.2 Example 1: Writing an item model

An average speed item has two variables, the speed `v` and the time `t`. Writing the distance as `v * t` keeps the answer a whole number. Each distractor is the answer of a misconception:

```
# Item models for a simulated Grade 7 field test.
speed <- item_model(
  stem = "A train travels {v * t} km in {t} hours. What is its average speed?",
  key = "{v} km/h",
  distractors = c(multiplied = "{v * t * t} km/h", subtracted = "{v * t - t} km/h",
                  added = "{v * t + t} km/h"),
  values = list(v = seq(40, 120, 10), t = 2:6), name = "SPD")
frac <- item_model(
  stem = "A class of {n} students voted. {num}/{den} of them chose soccer. How many
   ↪  students chose soccer?",
  key = "{n * num / den}",
  distractors = c(multiplied_only = "{n * num}", rest = "{n - n * num / den}",
                  one_part_too_many = "{n / den * (num + 1)}"),
  values = list(n = c(12, 18, 20, 24, 30, 36, 40), num = 1:3, den = 2:6),
  constraints = c("num < den", "n %% den == 0"), name = "FRA")
avg <- item_model(
  stem = "The heights of {k} plants are {paste(seq(a, by = s, length.out = k),
   ↪  collapse = ', ')} cm. What is their mean height?",
  key = "{a + s * (k - 1) / 2} cm",
```

```
  distractors = c(sum = "{k * a + s * k * (k - 1) / 2} cm", largest = "{a + s * (k -
   ↪  1)} cm",
                  divided_by_wrong_n = "{(k * a + s * k * (k - 1) / 2) / (k - 1)} cm"),
  values = list(k = 3:5, a = c(10, 12, 14, 20), s = c(2, 4)),
  constraints = "(s * (k - 1)) %% 2 == 0", name = "AVG")
speed
#> <mv_item_model> SPD
#>   Stem: A train travels {v * t} km in {t} hours. What is its average speed?
#>   Key:  {v} km/h
#>   Distractor 1: {v * t * t} km/h  (multiplied)
#>   Distractor 2: {v * t - t} km/h  (subtracted)
#>   Distractor 3: {v * t + t} km/h  (added)
#>   v: 40, 50, 60, ..., 110, 120
#>   t: 2, 3, 4, 5, 6
#>   45 combinations of values meet the constraints; 45 give items whose options
 ↪  all differ.
```

Printing a model checks it: All 45 combinations of speed and time give four different options. `generate_isomorphs()` draws items from the combinations and puts the options in random order:

```
x <- generate_isomorphs(speed, n = 3, seed = 15)
x$stem
#> [1] "A train travels 500 km in 5 hours. What is its average speed?"
#> [2] "A train travels 240 km in 6 hours. What is its average speed?"
#> [3] "A train travels 540 km in 6 hours. What is its average speed?"
as.data.frame(x[, c("id", "A", "B", "C", "D", "key")])
#>       id        A          B          C        D key
#> 1 SPD-01 100 km/h 2,500 km/h   505 km/h 495 km/h   A
#> 2 SPD-02 234 km/h    40 km/h 1,440 km/h 246 km/h   B
#> 3 SPD-03 546 km/h 3,240 km/h    90 km/h 534 km/h   C
```

The fraction model is less safe. With $\frac{1}{2}$, the students who chose soccer and those who did not are the same number, so the key and the `rest` distractor are identical. `generate_isomorphs()` leaves such combinations out, and says so:

```
fams <- generate_isomorphs(list(speed, frac, avg), n = 12, seed = 151)
#> FRA: 25 of 56 combinations of values give two identical options or cannot be
 ↪  computed, and were left out. Add a constraint to rule them out.
#> AVG: 2 of 24 combinations of values give two identical options or cannot be
 ↪  computed, and were left out. Add a constraint to rule them out.
table(fams$model)
#>
#> AVG FRA SPD
```

```
#>  12  12  12
```

Almost half of the fraction model's combinations were dropped, and two of the mean model's. The message is a request to fix the model with a constraint, so that the next person who uses it, or the software, does not depend on the check (Exercise 1).

15.3 Example 2: Calibrating the families

The 36 items were field-tested on 800 students. We simulate the responses: Each family has its own difficulty, each item adds item-specific noise with an SD of 0.2 logits, and fractions with a numerator above 1, which take two steps, are 0.8 logits harder than unit fractions. Then we calibrate the Rasch model.

```
# Simulated field test of the generated items: 800 students.
set.seed(1515)
two_step <- !is.na(fams$num) & fams$num > 1
b_true <- c(SPD = -0.2, FRA = -0.4, AVG = 0.4)[fams$model] + 0.8 * two_step +
  rnorm(nrow(fams), 0, 0.2)
theta <- rnorm(800)
resp <- sapply(b_true, function(b) rbinom(800, 1, plogis(theta - b)))
colnames(resp) <- fams$id
rasch <- mirt(as.data.frame(resp), 1, "Rasch", SE = TRUE, verbose = FALSE)
pars <- tidy(rasch)
items <- as.data.frame(fams)
b <- pars[pars$term == "b", ]
items$b <- b$estimate[match(items$id, b$item)]
items$b_se <- b$std.error[match(items$id, b$item)]
```

`calibrate_families()` summarizes each family. Its observed SD includes the estimation error of each b; with `se`, the error variance is subtracted, which leaves the within-family SD. The 95% interval is where a new isomorph's difficulty is expected to fall, and `field_n` translates the prediction error into the size of a field test that would estimate the item as precisely.

```
cal <- calibrate_families(items, family = "model", se = "b_se")
cal
#> Item families: 3 families, 36 items; SD within corrected for estimation error
#>  family items mean_b sd_within 95% for new item field_n isomorphic
#>     SPD    12  -0.25      0.19    -0.64 to 0.14     135        yes
#>     FRA    12   0.04      0.46    -0.89 to 0.97      23         NO
#>     AVG    12   0.51      0.27    -0.04 to 1.07      69        yes
#> field_n: Test takers a field test would need to estimate a new isomorph's b as
 ↪  precisely
#> as its family mean predicts it. Isomorphic: SD within at most 0.30.
```

The speed and mean families are isomorphic, with within-family SDs of about 0.2 and 0.3 logits: Their family means are worth a field test of 70 to 140 students. The fraction family spreads over almost two logits. Sorting its items by difficulty shows why:

```
fr <- items[items$model == "FRA", c("id", "num", "den", "b")]
fr[order(fr$b), ]
#>        id num den       b
#> 22 FRA-10   1   6 -0.6857
#> 18 FRA-06   1   5 -0.6532
#> 19 FRA-07   1   5 -0.4371
#> 15 FRA-03   1   4 -0.1901
#> 21 FRA-09   1   6 -0.0803
#> 17 FRA-05   3   4  0.0779
#> 24 FRA-12   2   6  0.1389
#> 13 FRA-01   2   3  0.1877
#> 23 FRA-11   2   6  0.4410
#> 20 FRA-08   3   5  0.4661
#> 16 FRA-04   3   4  0.5483
#> 14 FRA-02   2   3  0.6966
```

All five unit fractions are easier than all seven others. Splitting the family by that variable gives four isomorphic families (Figure 15.1):

```
items$family <- ifelse(items$model == "FRA",
                    ifelse(items$num == 1, "FRA-unit", "FRA-nonunit"), items$model)
cal2 <- calibrate_families(items, family = "family", se = "b_se")
cal2
#> Item families: 4 families, 36 items; SD within corrected for estimation error
#>       family items mean_b sd_within 95% for new item field_n isomorphic
#>     FRA-unit     5  -0.41      0.26    -0.96 to 0.15      76        yes
#>          SPD    12  -0.25      0.19    -0.64 to 0.14     135        yes
#>  FRA-nonunit     7   0.37      0.22    -0.09 to 0.82     106        yes
#>          AVG    12   0.51      0.27    -0.04 to 1.07      69        yes
#> field_n: Test takers a field test would need to estimate a new isomorph's b as
 ↪  precisely
#> as its family mean predicts it. Isomorphic: SD within at most 0.30.
```

```
autoplot(cal2)
```

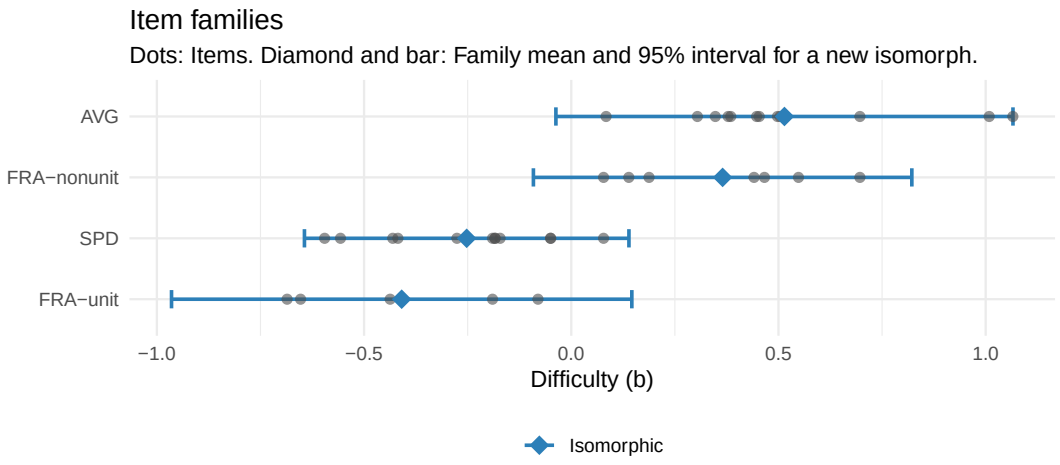


Figure 15.1: Item families after splitting the fraction model. Dots are items, diamonds family means, and bars the 95% interval for a new isomorph.

The split is a substantive finding, not a statistical repair: Finding $\frac{2}{3}$ of a number takes one step more than finding $\frac{1}{3}$ of it. The item writers can keep one model and treat `num > 1` as a feature, or write two models; either way, the family mean of the original model would have misled.

15.4 Example 3: One step, with a random item effect

The two-step approach treats the estimated difficulties as data. A one-step model fits the responses directly: The family is a fixed effect, and each item's departure from its family mean is a random effect whose variance is σ^2, pooled over families. mirt's `mixedmirt()` fits it (the estimation takes about a minute):

```
design <- data.frame(family = factor(items$family))
mm <- mixedmirt(as.data.frame(resp), model = 1, itemdesign = design,
                fixed = ~ 0 + family, random = ~ 1 | items, verbose = FALSE)
fixed <- summary(mm, verbose = FALSE)$fixed
data.frame(family = sub("family", "", rownames(fixed)),
           difficulty = -fixed[, "Estimate"])   # mirt models easiness; reverse the
            ↪  sign
#>         family difficulty
#> 1          AVG      0.503
#> 2 FRA-nonunit      0.357
#> 3    FRA-unit     -0.370
#> 4          SPD     -0.253
sqrt(coef(mm)$items)                                  # within-family SD
#>         COV_items_items
#> par               0.240
#> CI_2.5            0.145
```

```
#> CI_97.5           0.306
```

The family difficulties match the two-step means, and the within-family SD is about 0.24 logits, close to the 0.2 that was simulated. This is the LLTM of Chapter 14 with a random item effect, and the same model fits as a generalized linear mixed model with the eirm package (Bulut et al. 2021), from data in long format. Item features such as `num > 1` can go into the same formula:

```
# install.packages("eirm")
library(eirm)
long <- data.frame(person = rep(seq_len(nrow(resp)), ncol(resp)),
                   id = rep(colnames(resp), each = nrow(resp)),
                   resp = as.vector(resp))
long <- merge(long, items[, c("id", "model", "family")])
long$two_step <- as.integer(long$family == "FRA-nonunit")
m_eirm <- eirm("resp ~ -1 + model + two_step + (1|person) + (1|id)", data = long)
print(m_eirm, difficulty = TRUE)   # family difficulties and the two-step effect
lme4::VarCorr(m_eirm$model)       # person and within-family item variance
```

With the feature in the model, the original three item models stay, and the fixed effect of `two_step` estimates how much the second step adds. Hierarchical models of this kind can also give each family its own variance and its own discrimination (Sinharay et al. 2003; Geerlings et al. 2011).

15.5 Example 4: Does the family mean predict new items?

The real test of family calibration is a held-out one: Calibrate each family on some of its items and predict the others as if they were new.

```
number <- as.integer(sub(".*-", "", items$id))
old <- items[number <= 6, ]
new <- items[number > 6, ]
cal_old <- calibrate_families(old, family = "family", se = "b_se")
new$predicted <- cal_old$mean_b[match(new$family, cal_old$family)]
sqrt(mean((new$predicted - new$b)^2))
#> [1] 0.251
```

The family means miss the new items by about a quarter of a logit, about what the within-family SDs promised, and three times the standard error of an item calibrated on 800 students. That is precise enough to assemble field-test or practice forms, and to start online calibration, where new items are given as unscored pretest items and their own estimates replace the family mean as responses come in. Whether it is precise enough for scored items on a high-stakes test is a judgment about the decisions the scores support, not a property of the data.

15.6 Your turn

1. Add a constraint to `frac` so that no combination gives identical options, and print the model. Which fractions does your constraint rule out?
2. Without splitting the fraction family, fit `predict_difficulty(items, c("model", "num"), se = "b_se", methods = "lltm")`. What does the weight of `num` say, and how does its cross-validated error compare with that of the split families (`c("family")`)?
3. Widen the values of the speed model: Change `t` to `2:10` and `v` to `seq(5, 120, 5)`. Print the model. Are all items still flawless, and are they still isomorphs?
4. In Example 2, raise `max_sd` to 0.5. Which families now count as isomorphic, and what would that hide?

Solutions

```
# 1. a constraint for distinct options
frac2 <- item_model(
  stem = frac$stem, key = frac$key, distractors = frac$distractors, values =
   ↪  frac$values,
  constraints = c(frac$constraints,
                  "length(unique(c(n * num / den, n * num, n - n * num / den, n /
                   ↪  den * (num + 1)))) == 4"),
  name = "FRA")
frac2
#> <mv_item_model> FRA
#>   Stem: A class of {n} students voted. {num}/{den} of them chose soccer. How
 ↪  many students chose soccer?
#>   Key:  {n * num / den}
#>   Distractor 1: {n * num}  (multiplied only)
#>   Distractor 2: {n - n * num / den}  (rest)
#>   Distractor 3: {n / den * (num + 1)}  (one part too many)
#>   n: 12, 18, 20, ..., 36, 40
#>   num: 1, 2, 3
#>   den: 2, 3, 4, 5, 6
#>   Constraints: num < den; n %% den == 0; length(unique(c(n * num / den, n * num,
 ↪  n - n * num / den, n / den * (num + 1)))) == 4
#>   31 combinations of values meet the constraints; 31 give items whose options
 ↪  all differ.
g <- expand.grid(num = 1:3, den = 2:6)
g <- g[g$num < g$den, ]
g$distinct <- with(g, num / den != 1 / 2 & den != 2 * num + 1)
g[!g$distinct, ]
#>    num den distinct
#> 1    1   2    FALSE
#> 4    1   3    FALSE
```

```
#> 8    2   4    FALSE
#> 11   2   5    FALSE
#> 15   3   6    FALSE

# 2. the numerator as a feature
m_num <- predict_difficulty(items, c("model", "num"), se = "b_se", methods = "lltm")
tidy(m_num)
#> # A tibble: 5 x 4
#>   method term        estimate std_error
#>   <chr>  <chr>          <dbl>     <dbl>
#> 1 lltm   (Intercept)   -1.01     0.194
#> 2 lltm   model: AVG     0.753    0.110
#> 3 lltm   model: FRA     0.292    0.109
#> 4 lltm   model: SPD     0       NA
#> 5 lltm   num            0.415    0.0970
m_num$accuracy
#> # A tibble: 1 x 5
#>   method  rmse   mae     r    r2
#>   <chr>  <dbl> <dbl> <dbl> <dbl>
#> 1 lltm   0.276 0.214 0.791 0.625
predict_difficulty(items, "family", se = "b_se", methods = "lltm")$accuracy
#> # A tibble: 1 x 5
#>   method  rmse   mae     r    r2
#>   <chr>  <dbl> <dbl> <dbl> <dbl>
#> 1 lltm   0.249 0.205 0.833 0.694

# 3. wider values for the speed model
speed3 <- item_model(stem = speed$stem, key = speed$key, distractors =
 ↪  speed$distractors,
                     values = list(v = seq(5, 120, 5), t = 2:10), name = "SPD")
speed3
#> <mv_item_model> SPD
#>   Stem: A train travels {v * t} km in {t} hours. What is its average speed?
#>   Key:  {v} km/h
#>   Distractor 1: {v * t * t} km/h  (multiplied)
#>   Distractor 2: {v * t - t} km/h  (subtracted)
#>   Distractor 3: {v * t + t} km/h  (added)
#>   v: 5, 10, 15, ..., 115, 120
#>   t: 2, 3, 4, ..., 9, 10
#>   216 combinations of values meet the constraints; 216 give items whose options
 ↪  all differ.

# 4. a looser threshold
```

```
calibrate_families(items, family = "model", se = "b_se", max_sd = 0.5)
#> Item families: 3 families, 36 items; SD within corrected for estimation error
#>  family items mean_b sd_within 95% for new item field_n isomorphic
#>     SPD    12  -0.25      0.19    -0.64 to 0.14     135        yes
#>     FRA    12   0.04      0.46    -0.89 to 0.97      23        yes
#>     AVG    12   0.51      0.27    -0.04 to 1.07      69        yes
#> field_n: Test takers a field test would need to estimate a new isomorph's b as
 ↪  precisely
#> as its family mean predicts it. Isomorphic: SD within at most 0.50.
```

1. The constraint that the four answers are all different leaves 31 combinations. It rules out $\frac{1}{2}$ (also written $\frac{2}{4}$ and $\frac{3}{6}$), where the rest equals the key, and $\frac{1}{3}$ and $\frac{2}{5}$, where the rest equals one part too many. Writing the general constraint is safer than listing the fractions, because it also holds after someone adds new values.
2. Each unit of the numerator adds about 0.4 logits, and the LLTM's cross-validated error (about 0.28 logits) is close to that of the split families (0.25): The feature does most of the work of the split. It does a little worse because the simulated effect is a step from unit to non-unit fractions, which a straight line in `num` only approximates.
3. All 216 combinations still give four different options, so the check passes. But a distance of 10 km in 2 hours and one of 1,200 km in 10 hours are not the same task for a Grade 7 student: Larger numbers and less friendly divisions make items harder. Whether they are isomorphs is an empirical question that only a field test answers, and extending the values is a change to the item model that needs one.
4. With 0.5, the original fraction family (0.46) counts as isomorphic. Its interval for a new item then spans almost two logits: Every unit fraction item would be predicted about 0.45 logits too hard, and every other one about a third of a logit too easy. The threshold should come from how precise the family mean must be for its use, not be loosened to pass a family.

In the lab

> Lab 9, `run_lab("aig")`, writes an item model, calibrates a field test of 60 generated items, and finds the families that are not isomorphic, and why. The AI Hub collects the other AI tools in mirtverse.

Further reading

Gierl and Haladyna (2013) collect the theory and practice of AIG, and Gierl and Lai (2013) give a short introduction to item models. Glas and van der Linden (2003) and Sinharay, Johnson and Williamson (2003) model item families with random item parameters, and Geerlings, Glas and van der Linden (2011) add the rules used to generate the items as explanatory variables. Bulut, Gorgun and Yildirim-Erbasli (2021) show how to fit such explanatory models in R with

eirm.

Chapter 16

Automated scoring

Learning goals. After this chapter you can

- explain what an automated scoring engine learns from human scores, and why it is a **prediction model** of the raters rather than a measurement model;
- train a simple engine on human-scored text with `train_scorer()` and judge it on responses it has not seen;
- evaluate machine scores with exact and adjacent agreement, **quadratic weighted kappa**, the **standardized mean difference** and **PRMSE**, overall and within groups, using `agreement_report()`;
- recognize **construct-irrelevant** features that an engine can reward, and choose how much of the scoring an engine should do;
- place human raters and an engine on the **Wright map** with a many-facet Rasch model (`rater_facets()`), and see what machine scoring does to item thresholds and test scores (`scoring_shift()`).

Key terms: Automated scoring, PRMSE, feature, overfitting and cross-validation, human-in-the-loop, Wright map.

16.1 The ideas

Constructed responses, from a one-sentence explanation to a full essay, are scored by trained raters against a rubric. Chapter 8 treated the raters' scores as data for a polytomous IRT model and checked that two raters agree. Human scoring is slow and expensive, so many testing programs now use an **automated scoring engine**: A model that predicts the score a trained rater would give.

An engine is built the way any machine learning model is (Chapter 13). Each response is turned into **features**, such as the words it uses, its length, or its spelling, and a model learns from a set of human-scored responses which features go with which scores. The engine never reads the rubric. It learns whatever predicts the raters, including features the rubric does not

mention. Two consequences follow:

- **The engine can only be as good as its training scores.** If the raters are inconsistent, the engine learns their noise; if they are biased, it learns their bias.
- **Agreement is not the same as measuring the construct.** Long answers tend to earn high scores, so length predicts scores, but an engine that rewards length is scoring something other than the rubric. Such features add **construct-irrelevant** variance and invite gaming.

Machine scores are therefore judged against human scores before they are used, following Williamson, Xi and Breyer (2012):

- **Agreement** with a human rater: Exact and adjacent agreement and the quadratic weighted kappa (QWK), which should be at least 0.70 and not much below the agreement between two humans (degradation of at most 0.10).
- **Score level**: The standardized mean difference (SMD) between machine and human scores, within $\pm$0.15 overall.
- **Fairness**: The same SMD within groups of test takers, often within $\pm$0.10, because an engine can match humans overall and still score one group lower (Bridgeman et al. 2012).

Agreement with one rater undersells a good engine, because that rater makes errors too. The **proportional reduction in mean squared error** (PRMSE) asks a better question: How much of the variance of the *true* score, the average over many raters, do the machine scores explain (Loukina et al. 2020)? It uses the double-scored responses to estimate rater error and corrects for it. A PRMSE of 0.80 means the machine scores predict the true score about as well as a measure with reliability 0.80 would.

Finally, a program decides **how much** the engine does. It can be the **sole scorer**; a **check scorer** next to one human read, where a second person reads only the responses on which human and machine disagree; or not be used at all. The stronger the role, the stronger the evidence it needs (Yan et al. 2020).

16.2 Example 1: Training and judging an engine

A Grade 7 science test asks: *"Why does bread go mouldy faster on the kitchen counter than in the fridge?"* A full answer has three key ideas: mould is a living fungus, it grows from spores that are already on the bread or in the air, and warmth speeds its growth while cold slows it. Each answer scores 0 to 3, one point per idea. We simulate 900 answers, all scored by one rater and half of them by a second. Students state each idea in many different ways, and some state an idea wrongly with much the same words ("Cold makes it grow faster."), which earns no credit.

```
# Simulated short answers to the mouldy bread question, scored 0 to 3.
set.seed(1616)
pick <- function(...) sample(c(...), 1)
```

```
ideas <- list(   # one function per key idea; each call writes one sentence
  function() paste0(pick("Mould", "Mold", "The fuzz", "The green stuff", "It"), " ",
                    pick("is a fungus", "is a kind of fungus", "is a living fungus",
                         "is alive, a type of fungus", "is a living thing called a
                          ↪  fungus"), "."),
  function() paste0(pick("Its spores", "Tiny spores", "Mould spores", "Spores"), "
   ↪  ",
                    pick("are already on the bread", "float in the air",
                         "land on the bread from the air", "are everywhere in the
                          ↪  air"),
                    pick("", " and start to grow", " and grow on it"), "."),
  function() paste0(pick("Warmth", "A warm kitchen", "Heat", "Being warm"), " ",
                    pick("speeds up its growth", "makes it grow faster", "helps it
                     ↪  grow quickly"), " ",
                    pick("and the cold fridge slows it down", "while cold slows it",
                         "but low temperatures stop it growing fast"), "."))
wrong <- list(   # the same ideas stated wrongly, with much of the same vocabulary
  function() paste0(pick("Mould", "Mold", "The fuzz", "It"), " ",
                    pick("is not alive", "is a kind of plant", "is just dirt", "is
                     ↪  not a living thing"), "."),
  function() paste0(pick("The bread", "The flour", "Old bread"), " ",
                    pick("makes the mould by itself", "turns into mould", "grows the
                     ↪  fuzz on its own"), "."),
  function() paste0(pick("Cold", "The fridge", "A low temperature"), " ",
                    pick("makes it grow faster", "speeds up its growth", "helps
                     ↪  mould grow"), "."))
filler <- c("I think this is because of science.", "This happens a lot in my
 ↪  house.",
            "Bread is made of flour and water.", "It is gross when bread goes bad.")
answer <- function(theta) {
  has <- runif(3) < plogis(1.4 * (theta - c(-1, 0, 0.7)))
  bad <- !has & runif(3) < 0.4
  txt <- c(vapply(ideas[has], function(f) f(), ""), vapply(wrong[bad], function(f)
 ↪  f(), ""),
           sample(filler, rbinom(1, 2, 0.4)))
  if (!length(txt)) txt <- "I don't know."
  list(text = paste(sample(txt), collapse = " "), true = sum(has))
}
ans <- lapply(rnorm(900), answer)
rate <- function(true) pmin(3, pmax(0, round(true + rnorm(length(true), 0, 0.45))))
bread <- data.frame(id = 1:900, response = vapply(ans, `[[`, "", "text"),
                    true = vapply(ans, `[[`, 0, "true"))
bread$human1 <- rate(bread$true)
```

```
bread$human2 <- ifelse(runif(900) < 0.5, rate(bread$true), NA)
```

Two raters agree well on the double-scored answers. An engine has to do at least this well:

```
head(bread$response, 3)
#> [1] "A low temperature speeds up its growth. This happens a lot in my house. The
 ↪  fuzz is a living fungus."
#> [2] "Spores float in the air and grow on it. The green stuff is a kind of
 ↪  fungus."
#> [3] "Mold is a fungus. Being warm helps it grow quickly while cold slows it.
 ↪  This happens a lot in my house."
dbl <- !is.na(bread$human2)
agreement_report(bread$human2[dbl], bread$human1[dbl])[, c("n", "exact", "qwk")]
#> Machine-human agreement
#>    n exact  qwk
#>  458  0.66 0.81
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).
```

We train the engine on the first 600 answers and keep the last 300 as a test set that it never sees. `train_scorer()` fits a ridge regression of the first human score on the words and two-word phrases of each answer, and chooses the amount of shrinkage by cross-validation:

```
train <- bread[1:600, ]
test <- bread[601:900, ]
engine <- train_scorer(train, response = "response", score = "human1")
engine
#> <mv_scorer> ridge regression on words and two-word phrases
#>   600 responses, 366 features, lambda = 14.7, cross-validated RMSE = 0.51, QWK =
 ↪  0.87
#> Check it with agreement_report(); use it with predict() or score_open().
head(tidy(engine), 10)
#> # A tibble: 10 x 3
#>    by    term       weight
#>    <chr> <chr>       <dbl>
#>  1 all   fungus       0.451
#>  2 all   is a         0.294
#>  3 all   spores       0.226
#>  4 all   slows        0.207
#>  5 all   slows it     0.207
#>  6 all   the          0.197
#>  7 all   a fungus     0.190
#>  8 all   of fungus    0.172
#>  9 all   a            0.169
```

```
#> 10 all   cold slows  0.144
```

Among the words with the largest weights are the ones the rubric cares about, *fungus*, *spores* and *slows*. Nothing told the engine so; it learned them because raters gave points to answers that used them. But *is a* and *the* are near the top as well. They carry no idea; they go with longer answers, and longer answers tend to state more ideas. Now the test set:

```
test$machine <- predict(engine, test)
agreement_report(test$machine, test$human1, test$human2)
#> Machine-human agreement
#>  group   n n_double mean_human mean_machine exact adjacent  qwk qwk_human
#>    All 300      162       1.66         1.67  0.77     1.00 0.88      0.81
#>  corr  smd prmse verdict
#>  0.88 0.02  0.94   meets
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).
```

On answers it has never seen, the engine agrees with the first rater at least as well as the two raters agree with each other, its scores are at the same level (SMD near zero), and its PRMSE is above 0.9: It predicts the true score better than a single rater does. On this evidence it could at least serve as a check scorer. Figure 16.1 shows the same statistics as a picture.

```
autoplot(agreement_report(test$machine, test$human1, test$human2))
```

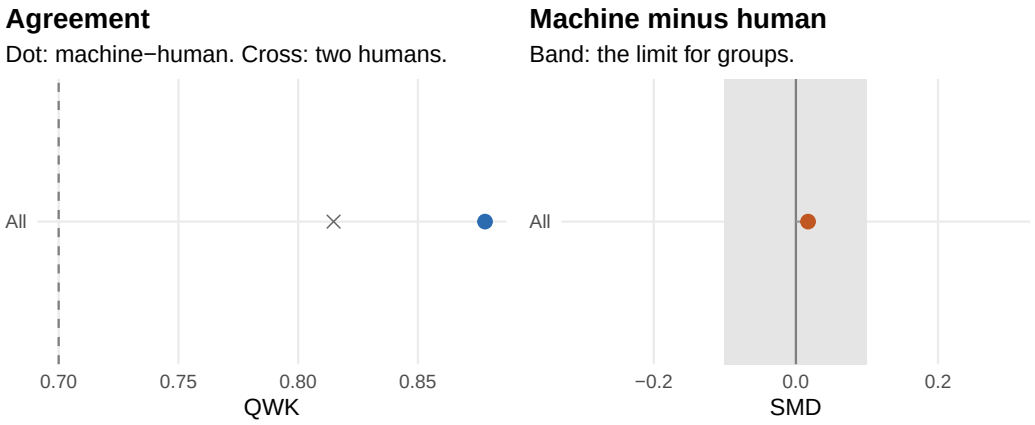


Figure 16.1: Agreement report for the engine on the test set: QWK with the first rater (dot) next to human-human QWK (cross), and the machine-human SMD with the band of ± 0.10.

16.3 Example 2: What is the engine rewarding?

Agreement statistics average over the answers in the test set. To see what the engine actually rewards, give it answers whose correct score you know. A rater would give the first two an-

swers below 0, because they name the right words but explain nothing; the third 0, because both its statements are wrong; and the fourth the full 3, because it states all three ideas in words the engine has rarely seen:

```
# Four answers with a known correct score, unlike any in the training data.
probe <- data.frame(response = c(
  "fungus spores cold warm fungus spores grow",
  "Spores spores spores. Fungus fungus fungus. Warm warm warm.",
  "The fuzz is not a living thing. Cold makes it grow faster.",
  "It is a mushroom-like organism whose seeds drift onto the loaf, and it
   ↪  multiplies quicker in heat than in a chilled fridge."),
  rater = c(0, 0, 0, 3))
```

```
probe$machine <- predict(engine, probe)
probe$raw <- round(predict(engine, probe, type = "raw"), 2)
probe[, c("rater", "machine", "raw")]
#>   rater machine  raw
#> 1     0       1 1.10
#> 2     0       1 1.21
#> 3     0       0 0.34
#> 4     3       1 1.08
```

The engine gets the third answer right: Wrong statements like these were common in the training data, and it learned that *not alive* and *grow faster* after *cold* go with low scores. It gets the other three wrong. It gives a point to each list of key words, which explains nothing, and only one point to the best answer, because that answer is phrased in words the training answers never used. Nothing in the agreement statistics of Example 1 warned about this: The test set held only answers like the training answers, so it could not show what happens with an unusual writer, or when test takers learn how the engine works.

This is the core lesson of automated scoring for a psychometrician: agreement with raters is necessary evidence, not sufficient evidence. A validity argument for machine scores also needs to show that the features the engine uses are part of the construct, that the engine treats groups of test takers the same way, and that the scoring design catches responses the engine was not built for. A check scorer design does the last part automatically: Answers on which human and machine disagree go to a second reader.

16.4 Example 3: The engine on the Wright map

Agreement statistics are on the score scale of one item. A testing program reports on the **logit scale** of the whole test, where items are banked and students are scored, and the Wright map (Chapter 7) shows that scale with students and items side by side. An engine can be put on it too, in two ways.

The first treats the engine as **one more rater**. A many-facet Rasch model (Linacre 1989) adds

a rater facet to the partial credit model,

$$\text{logit}\, P(X_{pir} = k \mid k-1 \text{ or } k) = \theta_p - \delta_{ik} - \lambda_r,$$

so each rater r, human or machine, gets a **severity** λ_r on the same scale as the students' θ_p and the thresholds δ_{ik}. `rater_facets()` fits it as an explanatory IRT model with the eirm package (Bulut et al. 2021). With `by_item = TRUE` each rater gets a severity for each item, the rater-by-item bias terms of a facets analysis.

The second asks what machine scores do to the **items**. `scoring_shift()` calibrates the same students twice, once with human and once with machine scores, links the two calibrations through items the engine does not score, and compares the thresholds. It also scores every student's machine-scored answers with the human calibration, which is what happens when a bank calibrated on human scores is used with an engine.

We simulate a science test of 600 students with 10 multiple-choice items and three written answers scored 0 to 3. A trained rater reads every answer, a second rater half of them, and an engine all of them. The engine misses one of the ideas on Q2, but you would not know that from its average over the three items.

```
# A test with 10 multiple-choice items and three written answers (Q1 to Q3),
# scored 0 to 3 by one or two raters and by an engine.
set.seed(1617)
n <- 600
theta <- rnorm(n)
mc <- data.frame(student = sprintf("S%03d", 1:n),
                 sapply(seq(-1.5, 1.5, length.out = 10), function(b)
                   as.integer(runif(n) < plogis(1.2 * (theta - b)))))
names(mc)[-1] <- sprintf("MC%02d", 1:10)
read <- function(true) pmin(3, pmax(0, true + sample(c(-1, 0, 0, 0, 0, 1),
 ↪  length(true), TRUE)))
written <- do.call(rbind, lapply(1:3, function(j) {
  true <- rowSums(sapply(c(-1, 0, 1) + c(-0.2, 0, 0.2)[j],
                         function(b) runif(n) < plogis(1.5 * (theta - b))))
  machine <- read(true)
  if (j == 2) {                       # the engine misses an idea on Q2
    miss <- machine > 0 & runif(n) < 0.35
    machine[miss] <- machine[miss] - 1
  }
  data.frame(student = mc$student, item = paste0("Q", j), human1 = read(true),
             human2 = ifelse(runif(n) < 0.5, read(true), NA), machine = machine)
}))
```

Each row of `written` is one answer with its three scores. Fit the facets model with a severity for each rater on each item:

```
fx <- rater_facets(written, raters = c("human1", "human2", "machine"), by_item =
 ↪  TRUE,
                   labels = c("Rater 1", "Rater 2", "Engine"))
fx
#> Rater facets: 3 raters (AI: Engine), 600 persons, 3 items, severity by item
#>
#> Rater severity (logits; positive = harsher than the average rater):
#>    rater ai severity   se
#>  Rater 1       -0.02 0.04
#>  Rater 2       -0.07 0.04
#>   Engine AI    +0.09 0.04
#>
#> Flags:
#>   [BLOCK] rater severity, Engine, Q2: +0.44 logits, SE 0.09.
#>     Engine is harsher than the human raters here.
```

Overall, the engine is only 0.1 logits harsher than the average rater, well within the range of the two people. Item by item it is not: On Q2 it is about 0.4 logits harsher than the human raters, several standard errors from zero. Figure 16.2 draws the map. The faint points in the rater column are the severities by item, and Q2's engine point sits apart.

```
gg_wright(fx)
```

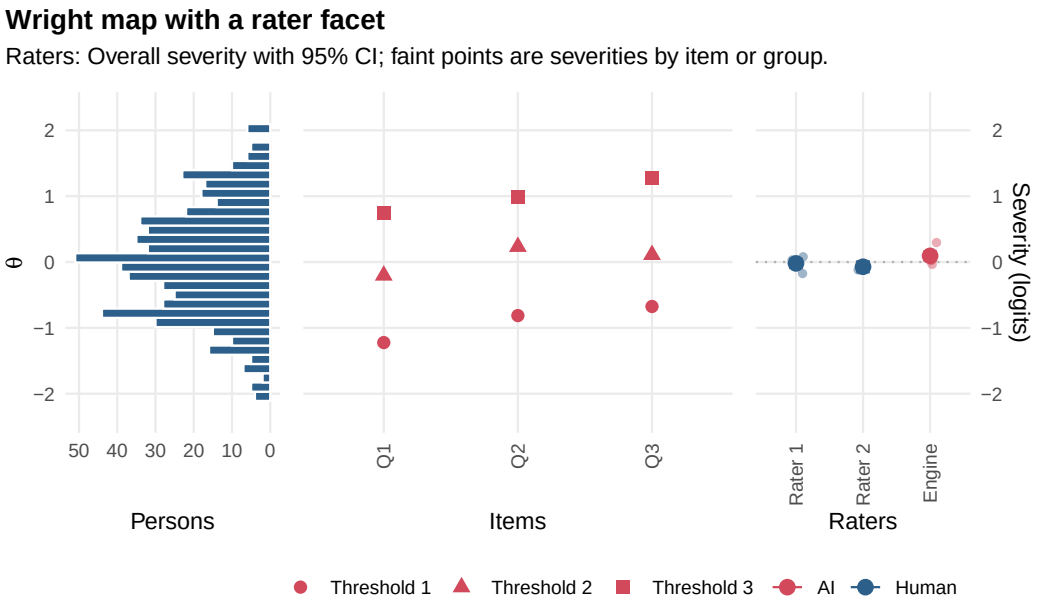


Figure 16.2: Wright map with a rater facet: Students (left), the thresholds of the three written answers (middle), and the severity of the two human raters and the engine (right), all on one logit scale. The faint points are each rater's severity on each item.

Now the items. Calibrate with human and with machine scores, linked through the multiple-choice items:

```
sh <- scoring_shift(written, human = "human1", anchors = mc)
sh
#> Human vs AI scoring: 3 AI-scored items, 600 persons, linked through 10 anchor
 ↪  items
#>
#> Threshold shift (machine minus human, logits):
#>  item a_human a_machine mean_shift max_shift
#>    Q1    1.62      1.59      -0.02     +0.12
#>    Q2    1.47      1.51      +0.46     +0.55
#>    Q3    1.55      1.59      -0.11     -0.23
#>
#> Score impact (mean theta change):
#>  group   n change   smd
#>    All 600  -0.04 -0.04
#>
#> Flags:
#>   [REVIEW] threshold shift, Q2, threshold 1: +0.55 logits (human
#>     -1.33, machine -0.78). This score is harder to reach when the
#>     engine scores: It is harsher here than human raters.
#>   [REVIEW] threshold shift, Q2, threshold 2: +0.38 logits (human
#>     0.05, machine 0.43). This score is harder to reach when the
#>     engine scores: It is harsher here than human raters.
#>   [REVIEW] threshold shift, Q2, threshold 3: +0.46 logits (human
#>     1.20, machine 1.66). This score is harder to reach when the
#>     engine scores: It is harsher here than human raters.
```

All three Q2 thresholds move up by about half a logit when the engine scores (Figure 16.3). A bank calibrated on machine scores would record Q2 as a harder item than it is, and a cut score set on it would move with it. The students' test scores move less, by about 0.04 SD, because Q2 is one item among 13. A small change in the total does not mean the engine is harmless: It is concentrated on one item, and in Lab 10 it is concentrated on one group of students.

```
gg_wright(sh)
```

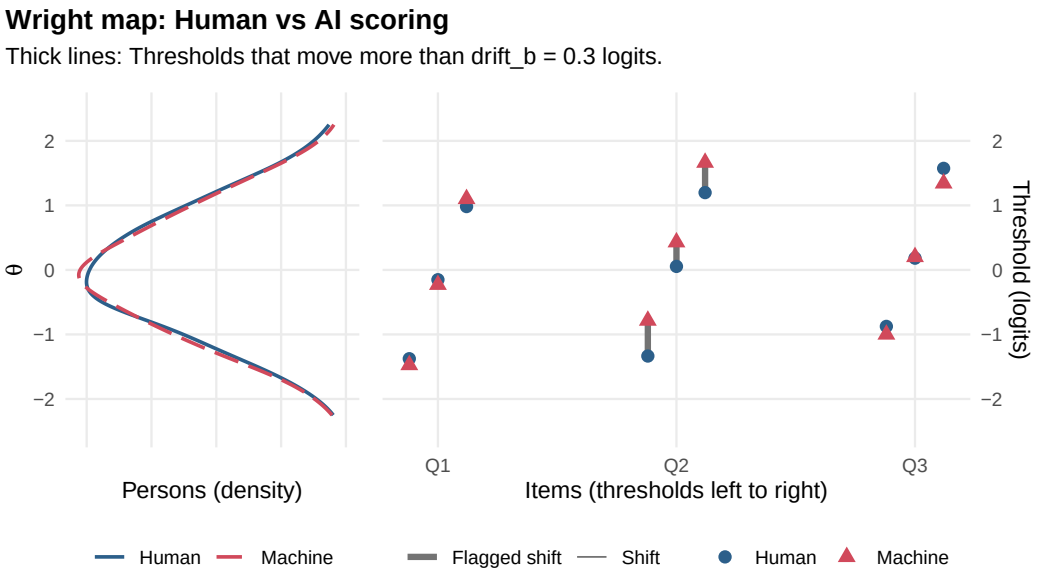


Figure 16.3: Wright map of human vs AI scoring: The students' scores (left) and the thresholds of each written answer (right) from human scores (circles) and machine scores (triangles). Thick lines mark thresholds that move by more than 0.3 logits.

Both results can join an audit: `ai_audit(components = list(fx, sh))` adds their flags to the checks of Chapter 18.

16.5 Your turn

1. Retrain the engine in Example 1 with `length = TRUE`, which adds the number of words as a feature. Does agreement on the test set change? Look at the weight of `(length)` in `tidy()`.
2. Train the engine on only the first 100 answers and evaluate it on the same test set. What happens to the QWK and the PRMSE, and why?
3. In Example 1, the engine was judged against `human1`, the score it was trained on. Compute the QWK of the machine scores with `human2` on the double-scored test answers. Is it lower or higher than with `human1`, and what does that tell you?
4. Your program is offered an engine with a QWK of 0.86 against one rater on a vendor's own test set. List three more pieces of evidence you would want before using it as the sole scorer.
5. In Example 3, fit `rater_facets()` without `by_item = TRUE`. Is the engine flagged? What does that tell you about checking an engine on average?

Solutions

```
# 1. adding response length
engine_len <- train_scorer(train, length = TRUE)
agreement_report(predict(engine_len, test), test$human1, test$human2)[, c("qwk",
 ↪  "smd", "prmse")]
#> Machine-human agreement
#>   qwk  smd prmse
#>  0.88 0.02  0.94
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).
tidy(engine_len)[tidy(engine_len)$term == "(length)", ]
#> # A tibble: 1 x 3
#>   by    term     weight
#>   <chr> <chr>     <dbl>
#> 1 all   (length) 0.0781

# 2. a small training set
engine_100 <- train_scorer(bread[1:100, ])
agreement_report(predict(engine_100, test), test$human1, test$human2)[, c("qwk",
 ↪  "smd", "prmse")]
#> Machine-human agreement
#>   qwk  smd prmse
#>  0.78 0.05  0.79
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).

# 3. agreement with the second rater
d <- !is.na(test$human2)
agreement_report(test$machine[d], test$human2[d])[, c("n", "qwk")]
#> Machine-human agreement
#>    n  qwk
#>  162 0.88
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).

# 5. one severity per rater
rater_facets(written, raters = c("human1", "human2", "machine"),
             labels = c("Rater 1", "Rater 2", "Engine"))
#> Rater facets: 3 raters (AI: Engine), 600 persons, 3 items
#>
#> Rater severity (logits; positive = harsher than the average rater):
#>    rater ai severity   se
#>  Rater 1       -0.02 0.04
#>  Rater 2       -0.07 0.04
```

```
#>   Engine AI    +0.09 0.04
#>
#> No AI rater differs from the human raters by more than 0.3 logits.
```

1. Agreement does not change, and length gets a small weight. Here the words already carry the key ideas, so length adds little. In real essays length is often one of the strongest predictors, which is why engines that use it need checks for gaming and for groups who write shorter answers.
2. With 100 answers the engine has seen fewer ways of stating each idea, so it misses more of them in new answers: QWK drops by about 0.1 and PRMSE to below 0.8, now no better than a single rater. Engines need hundreds of scored responses per item, and more for long answers.
3. The QWK with `human2` is about the same as with `human1`, although the engine never saw `human2`. The engine has learned the rubric's ideas, not the quirks of one rater; an engine that agreed much better with its training rater than with others would be copying that rater's errors.
4. For example: Agreement on an independent sample of your own test takers, not the vendor's; the SMD within groups such as English language learners (Bridgeman et al. 2012); PRMSE on double-scored responses; evidence that it does not reward length or key-word lists; and how unusual responses are routed to people.
5. No: Averaged over the three items, the engine is about 0.1 logits harsher than the people, inside the 0.3-logit limit. Its Q2 problem is diluted by two items it scores well. Check an engine item by item, and group by group, as `by_item = TRUE` and `group` do.

In the lab

> Lab 7, `run_lab("auto-scoring")`, trains an engine on 1,600 science explanations, checks it the way a testing program would, and asks whether it can score on its own, and places it on the Wright map next to the human readers. Lab 10, `run_lab("ai-audit")`, audits an engine that has been scoring a state test for two years and draws both Wright maps of Example 3. The AI Hub collects the other AI tools in mirtverse, and `score_open()` uses a trained engine, or a language model, inside the assessment pipeline.

Further reading

Williamson, Xi and Breyer (2012) set out the evaluation framework used in this chapter. Yan, Rupp and Foltz (2020) cover scoring engines and their operational use in depth. Loukina et al. (2020) explain PRMSE, and Bridgeman, Trapani and Attali (2012) show how an engine can differ from human raters for some groups of test takers. Linacre (1989) introduced the many-facet Rasch model for rater effects, and Bulut, Gorgun and Yildirim-Erbasli (2021) show how to fit it and other explanatory IRT models as generalized linear mixed models with the eirm package.

Chapter 17

Large language models as a colleague

Learning goals. After this chapter you can

- describe what a **large language model** (LLM) does and why its output varies and can be wrong;
- use an LLM through mirtverse to draft analysis code, write items and score responses, and see exactly what is sent to it;
- explain why every LLM output in testing needs a **check**, and use mirtverse's checks for wrong keys, weak distractors and machine-human agreement;
- make LLM work **reproducible** by recording and replaying responses, and say what to report about an LLM in a technical manual or paper.

Key terms: large language model, prompt, hallucination, human-in-the-loop, automatic item generation, automated scoring.

17.1 The ideas

A large language model is a neural network trained on a very large amount of text to predict the next piece of text. Given instructions and context, the **prompt**, it continues them, and that one ability is enough to write items, explain results, score essays against a rubric or draft code. Claude, GPT and the open models that run on a laptop through Ollama all work this way.

For a psychometrician, three properties matter more than how the network works:

- **Its output is a sample.** The same prompt can give different answers on different runs and with different versions of the model. An LLM is not a function with a fixed value; it behaves more like a rater.
- **It can be fluently wrong.** A model can write an item with no correct option, invent a reference or misread a rubric, in the same confident style as a correct answer. Such errors are called **hallucinations**, and they are found only by checking.
- **What you send leaves your computer.** Unless the model runs locally, the prompt goes to a provider. Test content and response data may be confidential, so what is sent has to

be a decision, not an accident.

These properties lead to the way mirtverse uses LLMs, which is also a good rule for any testing program (Hao et al. 2024; Bulut et al. 2024): The model drafts, something else checks, and a person decides. Generated items are answered again by a second, independent model call that does not see the key. Machine scores must agree with human scores, overall and within groups, before they count, following the framework of Williamson, Xi and Breyer (2012). Anything the model is unsure about goes to a person.

mirtverse talks to models through the ellmer package. `mv_ask_setup()` chooses a provider once: Claude or OpenAI with your own API key, or Ollama for a free model that runs on your own computer. This chapter instead runs in **replay mode**: `mv_llm_replay()` answers every request from a file of recorded responses that comes with mirtverse, so the examples need no key, no internet connection and give the same results for every reader. Those responses were written by hand to show what a model returns, including a few deliberate mistakes; they were not produced by a live model.

17.2 Example 1: From a sentence to a tested script

`mv_ask()` turns a plain-language request into an analysis plan and the plan into an R script built from tested templates. The model never writes code and never sees the responses: It sees the request and a summary of the columns. You can see that payload before anything is sent with `dry_run = TRUE`:

```
# mirt's Science data: 392 respondents, 4 attitude items scored 1 to 4.
resp <- Science
```

```
request <- "Fit a graded response model to these attitude items, check the fit and
 ↪  explain the results."
mv_ask(request, data = resp, dry_run = TRUE)
#> This would be sent to recorded responses:
#>
#> Request: Fit a graded response model to these attitude items, check the fit and
 ↪  explain the results.
#>
#> Data: 392 rows, 4 columns.
#> Columns (name | type | distinct values | min | max | share missing | guessed
 ↪  role | categories):
#> - Comfort | integer | 4 | 1 | 4 | 0.00 | item |
#> - Work | integer | 4 | 1 | 4 | 0.00 | item |
#> - Future | integer | 4 | 1 | 4 | 0.00 | item |
#> - Benefit | integer | 4 | 1 | 4 | 0.00 | item |
```

The payload holds the request, the number of rows and, for each column, its type, range, share missing and the role mirtverse guesses for it. With `anonymize = TRUE` the column names are

replaced by codes as well. Now the real call, answered from the recording:

```
s <- mv_ask(request, data = Science, ask = FALSE)
s$parser
#> [1] "recorded responses"
s$plan$model
#> [1] "graded"
s$plan$assumptions
#> [1] "All four columns (Comfort, Work, Future, Benefit) are items scored 1 to 4."
#> [2] "No grouping column was mentioned, so no DIF analysis is included."
#> [3] "Used the 4 columns that look like item responses as items."
```

The model chose the graded response model, kept the default steps, and stated two assumptions it made. Read them: An assumption is where a model fills a gap in your request, and a wrong one gives a wrong analysis that runs without error. The script itself is printed with `s`, and run with `run_script(s)`.

17.3 Example 2: Writing items, and catching the mistakes

The assessment pipeline writes items for the cells of a blueprint. Each multiple-choice item comes with a key and a rationale for every option, and is then checked by a second call that answers it blind.

```
# A two-cell blueprint for a short Grade 7 numeracy quiz.
cells <- data.frame(content = c("Ratios and rates", "Measurement"),
                    cognitive = c("Apply", "Apply"), n = c(2, 2))
```

```
p <- mv_pipeline("Grade 7 numeracy", verbose = FALSE)
p <- set_blueprint(p, cells, construct = "Grade 7 numeracy", use = "classroom
↪  diagnostic",
                   population = "Grade 7 students")
p <- suppressWarnings(generate_items(p, n = 2))
p$items[, c("id", "stem", "key", "verify_answer", "key_check")]
#> # A tibble: 4 x 5
#>   id    stem                                key   verify_answer key_check
#>   <chr> <chr>                               <chr> <chr>         <chr>
#> 1 G001  A recipe uses 3 cups of flour for e~ A     A             agree
#> 2 G002  A cyclist rides 45 km in 3 hours at~ C     C             agree
#> 3 G003  A garden is 12 m long and 7 m wide.~ A     B             disagree
#> 4 G004  A water bottle holds 2.5 litres. Ho~ B     B             agree
```

Two of the four items have problems. The writer keyed the garden item (G003) as 84 m, which is the garden's area, but the question asks for the fence around it, the perimeter of 38 m. The checking call answered B and the item is flagged. In the millilitre item (G004), one distractor

is a length, not a volume, so no test taker would choose it:

```
f <- flags(p, stage = "generation")
f[, c("check", "target", "level", "value")]
#> # A tibble: 4 x 4
#>   check      target                  level  value
#>   <chr>      <chr>                   <chr>  <chr>
#> 1 key_check  G003                    block  key A, verifier chose B; Fen~
#> 2 distractor G004                    review D (implausible)
#> 3 coverage   Ratios and rates / Apply review 2 items for a target of 2
#> 4 coverage   Measurement / Apply     review 2 items for a target of 2
p$items$verify_note[3]
#> [1] "Fencing around the garden is the perimeter, 2 x (12 + 7) = 38 m (option B);
 ↪  84 m is the area, so the key appears to be wrong."
```

A `block` flag stops the pipeline at the next stage until a person fixes the item or acknowledges the flag. The coverage flags are a reminder that items are lost in review and field testing, so a pool needs more items than the blueprint asks for. Notice what the checks do *not* do: They do not show that the items measure Grade 7 numeracy, that they are fair, or that their difficulty suits the test. Content review and a field test still do that. Items written one by one also have no family: Each needs its own field test. Template-based generation from item models (Chapter 15) trades variety for items that can be calibrated as a family.

17.4 Example 3: Is a recording reproducible?

Replay makes a session reproducible in a strong sense: The same calls on the same data give the same output, on any computer. It also makes the limits of a recording visible. Change the request by one word and the recording has no answer for it:

```
tryCatch(
  suppressWarnings(generate_items(set_blueprint(mv_pipeline(verbose = FALSE),
   ↪  c(Algebra = 2)),
                                  n = 1)),
  error = function(e) conditionMessage(e))
#> [1] "This request is not in the recording examples.json. A recording answers
 ↪  only the exact calls it was made from. Use a live model (mv_llm_replay(FALSE),
 ↪  then mv_ask_setup()), or record the call with mv_llm_record()."
```

For your own work, `mv_llm_record("session.json")` saves every request and response while you use a live model, and `mv_llm_replay("session.json")` plays them back. Keep the file with your analysis, as you would keep the data and the code. When you report work that used an LLM, say which model and version you used and when, give the prompts (mirtverse's are in its source code and in the recording), say what was sent and what was not, and describe how the outputs were checked.

Experimental: Language models as respondents. Some researchers ask a language model to answer items as a described person would ("a Grade 7 student who struggles with mathematics") and treat the answers as a "silicon sample" (Argyle et al. 2023). If that worked, items could be tried out before a field test. `simulate_llm_respondents()` lets you try it, and `compare_synthetic()` checks the result against real data. Here twelve personas answer one item from each of the twelve families in `itemtext`, whose field test gives the real proportions correct (`p_field`). The recorded answers were written to show the problems that studies of simulated respondents report:

```
items <- itemtext[endsWith(itemtext$id, "-01"), ]
personas <- c(
  "A Grade 7 student who struggles with mathematics.",
  "A Grade 7 student who struggles with mathematics and is still learning English.",
  "A Grade 7 student who finds fractions and decimals hard but tries every
   ↪  question.",
  "A Grade 7 student who often rushes and makes careless mistakes.",
  "A Grade 7 student who is about average in mathematics.",
  "A Grade 7 student who is about average in mathematics and is still learning
   ↪  English.",
  "A Grade 7 student with average marks who is anxious on tests.",
  "A Grade 7 student who is about average in mathematics and likes word problems.",
  "A Grade 7 student who is strong in mathematics.",
  "A Grade 7 student who is strong in mathematics and is still learning English.",
  "A Grade 7 student in an enriched mathematics program.",
  "A Grade 7 student who is strong in mathematics but reads slowly.")
syn <- simulate_llm_respondents(items, personas)
compare_synthetic(syn, setNames(items$p_field, items$id))
#> <mv_synthetic_comparison> 12 simulated respondents, 12 items (EXPERIMENTAL)
#>    item p_real p_synthetic difference
#>  N01-01   0.54        1.00       0.46
#>  N02-01   0.34        1.00       0.66
#>  N03-01   0.62        1.00       0.38
#>  N04-01   0.41        0.42       0.01
#>  N05-01   0.62        1.00       0.38
#>  N06-01   0.39        0.92       0.53
#>  N07-01   0.22        0.67       0.45
#>  N08-01   0.28        0.17      -0.11
#>  N09-01   0.32        0.92       0.60
#>  N10-01   0.62        1.00       0.38
#>  N11-01   0.36        0.17      -0.19
#>  N12-01   0.58        1.00       0.42
#>
#> * The simulated respondents find the items easier than real test
#>   takers: Mean proportion correct 0.77 against 0.44.
```

```
#> * They order the items by difficulty only loosely like real test
#>   takers do (Spearman correlation 0.68), so they cannot stand in
#>   for a field test of difficulty.
#> * Their answers vary little: 5 distinct response patterns among
#>   12 simulated respondents. Respondents described alike answer
#>   alike.
#> * Simulated respondents are experimental: Use them to find
#>   unclear items or to plan a study, not to estimate item
#>   parameters or to replace a field test.
```

Even "struggling" personas get most items right, so the items look far easier than they are. The short items, such as adding fractions (N06-01) and comparing decimals (N09-01), look easy to the model whatever their real difficulty, and the long word problems look hard. Personas described alike answer alike, and the ones described as "still learning English" miss every long word problem whatever their mathematics: A stereotype, not a measurement. Simulated respondents can help find unclear wording or plan a study. They cannot estimate item parameters, check fairness or replace a field test, and mirtverse labels them experimental for that reason.

17.5 Where do synthetic respondents land?

`compare_synthetic()` compares summaries, such as proportions correct. Summaries can look acceptable while most of the real population is missing from the synthetic sample. The coverage strip shows where synthetic respondents land. `synthetic_coverage()` cuts the real score distribution into ten slices of equal size (deciles) and counts how many synthetic respondents fall into each, and `gg_coverage()` draws one row per attempt, so you can see whether a pattern repeats.

mirtverse includes the data of a study of synthetic personality data: `hexaco_real` holds the answers of 17,874 real people from Canada, Great Britain and the United States to the 240 IPIP-HEXACO items (Ashton et al. 2007), and `hexaco_llm` holds 59,611 synthetic respondents from four open-weight language models, generated for the same countries in ten attempts each. To keep the example small, we use the 40 Honesty-Humility items, Canada only, and the first four attempts. Answers are stored as given, so the reverse-keyed items are reversed first:

```
h <- hexaco_items$item[hexaco_items$trait == "H"]       # 40 items
rev_items <- intersect(h, hexaco_items$item[hexaco_items$reverse])
keyed <- function(d) {
  d <- d[h]
  d[rev_items] <- 8 - d[rev_items]
  d
}
real <- keyed(hexaco_real[hexaco_real$country == "Canada", ])
```

```
ca <- hexaco_llm[hexaco_llm$country == "Canada" & hexaco_llm$attempt <= 4, ]
# One list per model, with one data frame per attempt
synthetic <- lapply(split(ca, ca$model),
                    function(m) lapply(split(m, m$attempt), keyed))
```

```
cov <- synthetic_coverage(synthetic, real, cor = "pearson")
gg_coverage(cov)
```

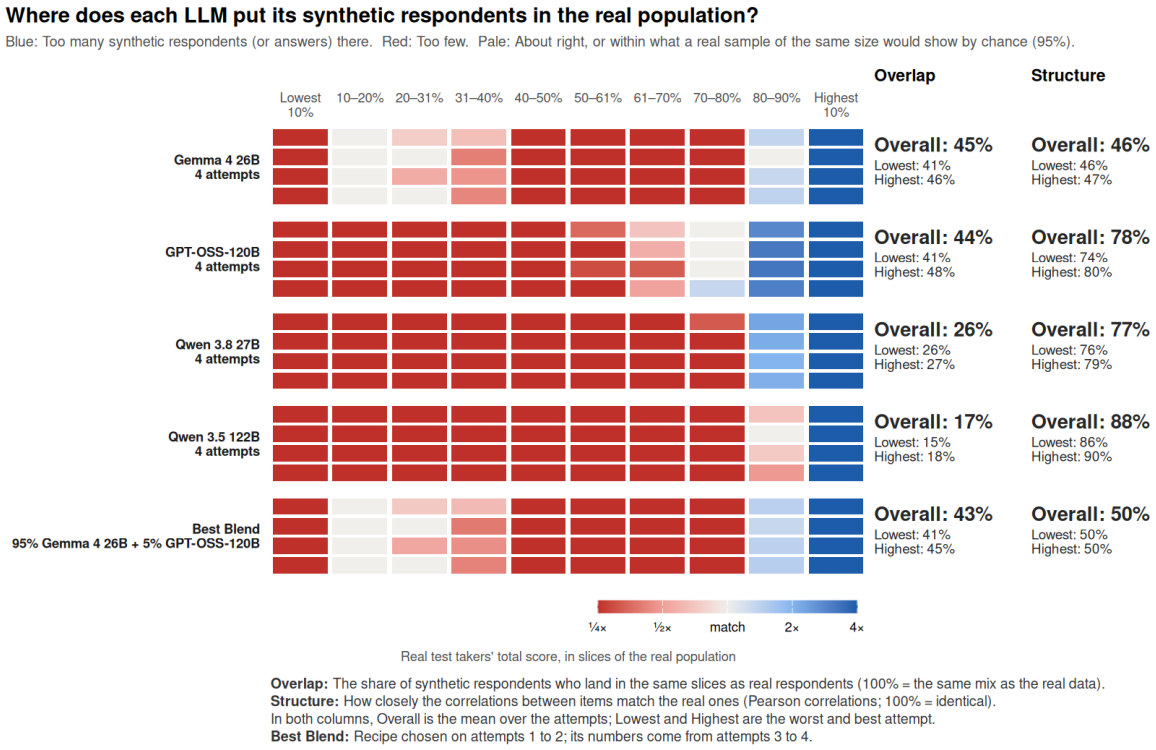


Figure 17.1: Coverage strip for Honesty-Humility in Canada: Four language models, four attempts each.

Columns are deciles of the real Honesty-Humility score, from the lowest 10% on the left to the highest 10% on the right. Blue cells hold too many synthetic respondents, red cells too few, and pale cells are within what a real sample of the same size would show by chance. The two numbers on the right summarize each model:

- **Overlap** is the share of synthetic respondents who land in the same deciles as real respondents. 100% means the same mix as the real population; a sample that puts everyone into one decile scores about 10%.
- **Structure** is how closely the correlations between items match the real ones (100% means identical).

Every model puts far too many synthetic Canadians in the top decile and too few in most other deciles: The synthetic people are more honest and modest than real people, and more alike.

The rows within each model look the same, so this is how the models behave, not one unlucky run. Note also that the model with the best Structure has the worst Overlap. A model can know which items go together and still put its respondents in the wrong place, which is why checks based only on correlations, such as reliability or factor structure, are not enough. The article Checking synthetic respondents shows the other options of the strip: Groups such as countries, single items and their answer categories, and the Best Blend, a mix of models chosen on some attempts and checked on others.

17.6 Your turn

1. Run `mv_ask(request, data = Science, dry_run = TRUE, anonymize = TRUE)`. What is still sent? Is anything in it confidential for your own data?
2. List the recordings that come with mirtverse with `mv_llm_recordings()`. Which mirtverse functions made them?
3. In Example 2, fix the miskeyed item with `p$items$key[3] <- "B"` and acknowledge the flag with `acknowledge()`. Why does mirtverse ask for a person to acknowledge a flag rather than clear it automatically?
4. Your program wants an LLM to score 20,000 short answers next spring. Name three pieces of evidence you would ask for before the machine scores count, and one group of test takers you would check separately.

Solutions

```
mv_ask(request, data = Science, dry_run = TRUE, anonymize = TRUE)
#> This would be sent to recorded responses:
#>
#> Request: Fit a graded response model to these attitude items, check the fit and
 ↪  explain the results.
#>
#> Data: 392 rows, 4 columns.
#> Columns (name | type | distinct values | min | max | share missing | guessed
 ↪  role | categories):
#> - V1 | integer | 4 | 1 | 4 | 0.00 | item |
#> - V2 | integer | 4 | 1 | 4 | 0.00 | item |
#> - V3 | integer | 4 | 1 | 4 | 0.00 | item |
#> - V4 | integer | 4 | 1 | 4 | 0.00 | item |
mv_llm_recordings()[, c("function_name", "prompt_start")]
#> # A tibble: 26 x 2
#>   function_name  prompt_start
#>   <chr>          <chr>
#> 1 mv_ask         "Request: Fit a graded response model to these attitude ~
#> 2 generate_items "Construct: Grade 7 numeracy Test takers: Grade 7 studen~
#> 3 generate_items "Construct: Grade 7 numeracy Test takers: Grade 7 studen~
```

```
#> 4 verify_items   "A recipe uses 3 cups of flour for every 2 cups of milk.~
#> 5 verify_items   "A cyclist rides 45 km in 3 hours at a steady speed. How~
#> 6 verify_items   "A garden is 12 m long and 7 m wide. How many metres of ~
#> # i 20 more rows
```

1. The request (with column names coded), the number of rows and the summary of each column (type, distinct values, range, share missing, guessed role) are still sent. Category labels are left out. Individual responses are never sent. For a confidential test, the request itself or the column names may give away content, which is why `anonymize = TRUE` exists; a local model through Ollama sends nothing at all.
2. `mv_ask()`, `generate_items()` (twice, one call per blueprint cell), `verify_items()` (once per item, called by `generate_items()`) and `score_open()` (once per batch of 10 responses, from the example in `system.file("replay", "examples.R", package = "mirtverse")`) and `simulate_llm_respondents()` (once per persona, from the box on simulated respondents).
3. A check can only say that something looks wrong; deciding whether it is, and what to do, is a professional judgement. Recording who acknowledged which flag, and why, is part of the evidence for the test.
4. For example: Agreement between machine and human scores on a sample of double-scored responses, compared with agreement between two humans (Williamson et al. 2012); the standardized mean difference between machine and human scores; and the same statistics within groups, such as English language learners, whose answers may be shorter or phrased differently at the same level of understanding. `score_open()` reports all of these.

In the AI Hub

> The AI Hub shows the AI features of mirtverse in one place. Lab 7, `run_lab("auto-scoring")`, trains and checks an automated scoring engine; Lab 8, `run_lab("item-text")`, predicts item difficulty from the text; Lab 9, `run_lab("aig")`, generates items from item models and calibrates their families; Lab 10, `run_lab("ai-audit")`, audits an AI-scored test before its scores are released; and the mv_ask() guide explains what `mv_ask()` sends and how it was benchmarked.

Further reading

Hao et al. (2024) describe what large language models change in testing, from item development to score reporting. Bulut et al. (2024) discuss the opportunities and the ethical questions, and Williamson, Xi and Breyer (2012) remain the reference for evaluating automated scores.

Chapter 18

Validity, fairness and governance of AI

Learning goals. After this chapter you can

- place an AI component in the **argument-based approach** to validity, and name the threats it adds to each inference;
- audit an automated scorer with `ai_audit()`: Agreement with human raters, **differential scoring** across groups, **drift** across administrations and DIF on the test as scored;
- fold the results of difficulty models and item generators into the same audit;
- draft a validity argument with `validity_argument()`, and say what evidence no statistic can provide: Human oversight, disclosure and appeals.

Key terms: Validity argument, differential scoring, DIF, human-in-the-loop, automated scoring.

18.1 The ideas

The earlier chapters of this part each checked one AI component on its own terms: A scoring engine against human raters (Chapter 16), a difficulty model by cross-validation (Chapter 14), an item generator by calibrating its families (Chapter 15). A testing program needs one more step: To ask what the components do to the **validity** of the scores, for the use the scores are put to. The *Standards* (American Educational Research Association et al. 2014) apply to AI components as to any other part of a test, and the guidelines of the International Test Commission and the Association of Test Publishers for technology-based assessment (International Test Commission and Association of Test Publishers 2022) add advice on automated scoring, security and accessibility.

The **argument-based approach** (Kane 2013) organizes the evidence. An interpretation-and-use argument is a chain of inferences from a response to a decision, each resting on a claim that evidence must support:

- **Scoring:** The score follows the rubric as trained raters apply it.

- **Generalization:** The score would be about the same with another occasion, form, rater or engine version.
- **Extrapolation:** The score reflects the construct, equally for all groups of test takers.
- **Decision:** Using the score for its purpose is appropriate.

An AI component adds threats to every link. A scoring engine may reward features that are not in the rubric, such as length or vocabulary (Williamson et al. 2012). It may be retrained or updated by a vendor between administrations, so that a score means something different from one year to the next. It may treat groups differently: Answers that people would score the same may get lower machine scores when they are typed on a phone or written by English language learners. Johnson, Liu and McCaffrey (2022) separate two questions here. **Measurement bias** asks whether the scores measure the construct equally for all groups, which is the familiar DIF question. **Algorithmic bias** asks whether the engine predicts the human score equally well for all groups, given the human score. The second is checked by comparing machine and human scores within groups among responses that people scored the same; we call a failure of that check **differential scoring**.

The decision inference is different in kind. It is supported by how the scores are used: Which responses a person reads before a score counts (**human-in-the-loop**), what test takers are told about automated scoring, how a score can be appealed, and how the component is monitored after it goes live. These are matters of policy and documentation. An audit can show which responses need a person; it cannot show that a person reads them.

18.2 Example 1: Auditing a scoring engine

A college uses an automated engine to score two writing prompts, W1 and W2, on a 0 to 4 rubric, for placement into first-year writing courses. Test takers write in a spring or a fall window, on a laptop or a phone. A validity sample of 30% is also scored by a person, and half of those by a second person. We simulate the scores. The engine rewards length, and answers typed on a phone are shorter, so on W2 it gives some phone answers a point less than their true score. A stricter engine version was installed for the fall window, and it marks down some W1 answers.

```
# Simulated machine and human scores of two writing prompts, 2,400 test takers.
set.seed(1818)
n <- 2400
writing <- data.frame(
  person = rep(sprintf("P%04d", 1:n), each = 2),
  window = rep(rep(c("Spring", "Fall"), each = n / 2), each = 2),
  device = rep(ifelse(runif(n) < 0.3, "Phone", "Laptop"), each = 2),
  prompt = rep(c("W1", "W2"), n))
theta <- rep(rnorm(n), each = 2)
true <- pmin(pmax(round(2 + 1.1 * theta + rnorm(2 * n, 0, 0.5)), 0), 4)
noisy <- function(x, p)        # off by one point with probability p
  pmin(pmax(x + ifelse(runif(length(x)) < p, sample(c(-1, 1), length(x), TRUE), 0),
   ↪  0), 4)
```

```
writing$machine <- noisy(true, 0.25)
short <- writing$device == "Phone" & writing$prompt == "W2" & true > 1 & runif(2 *
 ↪  n) < 0.35
writing$machine[short] <- true[short] - 1          # shorter answers marked down
strict <- writing$window == "Fall" & writing$prompt == "W1" & true > 0 & runif(2 *
 ↪  n) < 0.30
writing$machine[strict] <- true[strict] - 1        # the fall engine version
read <- rep(runif(n) < 0.3, each = 2)
twice <- read & rep(runif(n) < 0.5, each = 2)
writing$human1 <- ifelse(read, noisy(true, 0.25), NA)
writing$human2 <- ifelse(twice, noisy(true, 0.25), NA)
writing$device <- factor(writing$device, levels = c("Laptop", "Phone"))
writing$window <- factor(writing$window, levels = c("Spring", "Fall"))
```

The usual first check is agreement with human raters, which `agreement_report()` gives (Chapter 16):

```
r <- writing[!is.na(writing$human1), ]
agreement_report(r$machine, r$human1, r$human2, by = r$prompt)
#> Machine-human agreement
#>  by group   n n_double mean_human mean_machine exact adjacent  qwk
#>  W1   All 747      369       1.95         1.82  0.60     0.97 0.83
#>  W2   All 747      369       1.98         1.94  0.61     0.98 0.84
#>  qwk_human corr   smd prmse verdict
#>       0.86 0.83 -0.11  0.78   meets
#>       0.85 0.85 -0.04  0.80   meets
#> Meets: QWK >= 0.7 and within 0.1 of human-human QWK; |SMD| <= 0.15 overall and
 ↪  <= 0.1 within groups (mv_rules()).
```

Both prompts meet the rules. `ai_audit()` runs this check and the others in one call. The first group of `device` (Laptop) is the reference group, and the first `window` (Spring) is the baseline for drift:

```
audit <- ai_audit(writing, human2 = "human2", group = "device", item = "prompt",
                  period = "window")
audit
#> AI audit: 6 checks, 2 flags (2 block, 0 review)
#>                 check tested flagged status
#>       human agreement      2       0     ok
#>     machine agreement      2       0     ok
#>          overall bias      2       0     ok
#>    subgroup agreement      4       0     ok
#>  differential scoring      2       1  BLOCK
#>                 drift      2       1  BLOCK
```

```
#>
#> Flags:
#>   [BLOCK] differential scoring, W2, Phone: gap -0.19 SD, SE 0.04.
#>     For responses human raters scored the same, the engine scores
#>     Phone students lower than Laptop students.
#>   [BLOCK] drift, W1, Fall: SMD -0.20 against -0.04, QWK 0.79
#>     against 0.86. The engine's scores have changed since the
#>     first period: Check what changed (engine version, prompts,
#>     population) and rescore if needed.
```

Agreement passes overall and within groups, but two checks block. The differential scoring check compares the machine-minus-human difference of phone and laptop answers that people scored the same:

```
audit$differential
#> # A tibble: 2 x 6
#>   item  group reference     n      gap     se
#>   <chr> <chr> <chr>     <int>    <dbl>  <dbl>
#> 1 W1    Phone Laptop      217 -0.00220 0.0439
#> 2 W2    Phone Laptop      217 -0.195   0.0441
```

On W2, the engine scores phone answers about 0.2 SD lower than laptop answers that people scored the same, four to five standard errors from zero. This is what agreement within groups can miss: Phone users may also score lower in truth (here they do not), so a lower mean alone proves nothing, while a gap at the same human score is the engine's. The drift check compares each window with the first:

```
audit$drift[, c("item", "period", "n", "smd", "qwk", "smd_change", "p_change",
 ↪  "drift")]
#> # A tibble: 4 x 8
#>   item  period     n     smd   qwk smd_change  p_change drift
#>   <chr> <chr>  <dbl>   <dbl> <dbl>      <dbl>     <dbl> <lgl>
#> 1 W1    Spring   400 -0.0418 0.863    NA      NA        FALSE
#> 2 W1    Fall     347 -0.200  0.788    -0.159   0.000391 TRUE
#> 3 W2    Spring   400 -0.0451 0.859    NA      NA        FALSE
#> 4 W2    Fall     347 -0.0349 0.825     0.0102  1        FALSE
```

In the fall, W1's machine scores fell to 0.2 SD below the human scores, and agreement dropped. A drift flag needs both a change larger than the rule and a change beyond sampling error; `p_change` is Bonferroni-adjusted across the items and windows tested. Figure 18.1 shows the three score checks.

```
autoplot(audit)
```

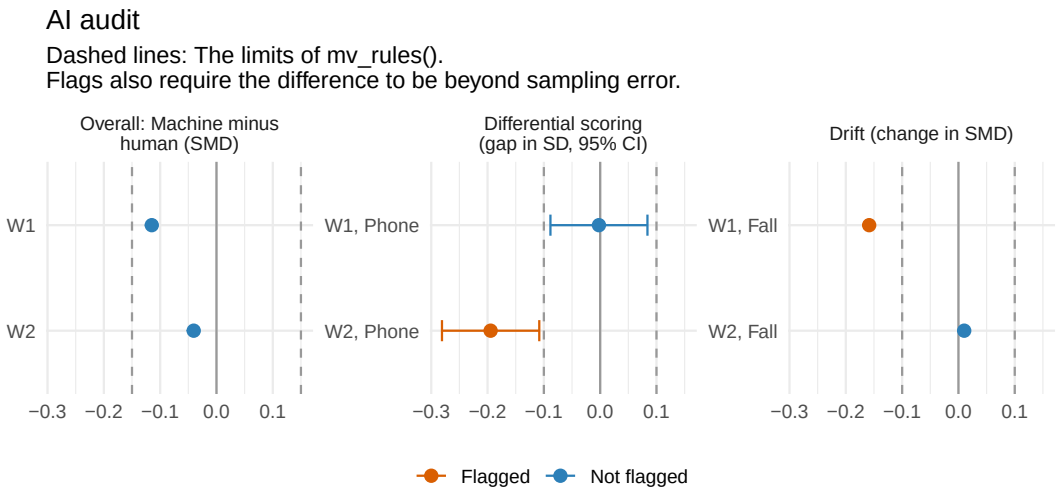


Figure 18.1: The score checks of the audit. Dashed lines are the default limits; bars are 95% confidence intervals.

When the test as scored is available, `responses` adds a DIF analysis with `dif()`. DIF on an AI-scored item is flagged for review; DIF on an item the engine does not score is reported for information, because it belongs to content review, not to the engine. Lab 10 does this.

18.3 Example 2: Other AI components

The same audit takes the results of the tools of the earlier chapters. Here a difficulty model was trained with the proportion correct of the field test among its features, and one of three item families is not isomorphic:

```
# Simulated field-test results of 45 items from three item models.
set.seed(181)
fam <- rep(c("Ratios", "Fractions", "Equations"), each = 15)
b <- c(Ratios = -0.8, Fractions = 0, Equations = 0.9)[fam] +
  rnorm(45, 0, rep(c(0.1, 0.1, 0.5), each = 15))
items <- data.frame(id = sprintf("I%02d", 1:45), family = fam, b = b, b_se = 0.08,
                    words = round(25 + 5 * b + rnorm(45, 0, 6)),
                    p_field = round(plogis(-1.2 * b + rnorm(45, 0, 0.05)), 3))
```

```
model <- predict_difficulty(items, c("family", "words", "p_field"), se = "b_se",
                            methods = "lltm")
families <- calibrate_families(items, family = "family", se = "b_se")
ai_audit(components = list(model, families))
#> AI audit: 2 checks, 2 flags (1 block, 1 review)
```

```
#>        check tested flagged status
#>      leakage      3       1  BLOCK
#>  isomorphism      3       1 REVIEW
#>
#> Flags:
#>   [BLOCK] leakage, p_field: rank correlation with b -0.99. A
#>     feature this close to the label is probably computed from the
#>     same responses: Leave it out.
#>   [REVIEW] isomorphism, Equations: SD within 0.46. The family
#>     mean is not a good estimate for new items from this item
#>     model.
```

The leaked feature blocks: Its predictions would vanish for new items that have no field test (Chapter 14). The family that is not isomorphic is flagged for review: Its mean is a poor stand-in for new items' difficulties (Chapter 15).

18.4 Example 3: From audit to argument

`validity_argument()` turns the audit into a draft argument. Each inference gets its claim, the threats an AI component adds, the evidence from the audit and a status:

```
va <- validity_argument(audit, construct = "academic writing in English",
                        use = "placement into first-year writing courses",
                        population = "incoming students")
va[, c("inference", "status")]
#> # A tibble: 4 x 2
#>   inference      status
#>   <chr>          <chr>
#> 1 Scoring        supported
#> 2 Generalization threatened
#> 3 Extrapolation  threatened
#> 4 Decision       needs evidence
```

```
cat(format(va, "markdown"))
#> The argument for using academic writing in English scores for placement into
 ↪  first-year writing courses, with automated scoring of constructed responses, by
 ↪  the four inferences of the argument-based approach (Kane, 2013).
#>
#> ### Scoring (supported)
#>
#> - **Claim:** Scores from automated scoring of constructed responses follow the
 ↪  scoring rubric as trained human raters apply it.
#> - **Threats:** The engine rewards surface features (length, vocabulary,
 ↪  spelling) rather than the ideas in the rubric; human scores used to train or
 ↪  check it are unreliable.
```

```
#> - **Evidence:** Human agreement: 2 tested, no flags. Machine agreement: 2
 ↪  tested, no flags. Overall bias: 2 tested, no flags.
#>
#> ### Generalization (threatened)
#>
#> - **Claim:** Scores would be the same on another occasion, form, rater or engine
 ↪  version, so they mean the same across administrations.
#> - **Threats:** An engine update, a new prompt or a change in the population
 ↪  shifts machine scores; agreement holds in one year only.
#> - **Evidence:** Drift: 2 tested, flagged W1, Fall [SMD -0.20 against -0.04, QWK
 ↪  0.79 against 0.86].
#> - **To do:** Resolve the flags, then rerun the audit before the scores are used.
#>
#> ### Extrapolation (threatened)
#>
#> - **Claim:** Scores reflect academic writing in English, and do so equally for
 ↪  all groups of incoming students.
#> - **Threats:** Features unrelated to the construct, such as language
 ↪  proficiency, lower machine scores for some groups; DIF on AI-scored items.
#> - **Evidence:** Subgroup agreement: 4 tested, no flags. Differential scoring: 2
 ↪  tested, flagged W2, Phone [gap -0.19 SD, SE 0.04].
#> - **To do:** Resolve the flags, then rerun the audit before the scores are used.
 ↪  Not checked yet: DIF.
#>
#> ### Decision (needs evidence)
#>
#> - **Claim:** Using the scores for placement into first-year writing courses is
 ↪  appropriate, with the human oversight, disclosure and appeals the use requires.
#> - **Threats:** Machine scores are used without human review where stakes are
 ↪  high; test takers are not told how they were scored; errors cannot be appealed.
#> - **Evidence:** None yet.
#> - **To do:** Document human review of flagged and low-confidence responses,
 ↪  disclosure to test takers, and an appeals process. No audit can supply this
 ↪  evidence.
```

The scoring inference is supported: The engine agrees with people as well as they agree with each other. Generalization and extrapolation are threatened by the fall engine version and by the phone gap. The decision inference **needs evidence** whatever the audit finds, and the draft says what: Human review of flagged responses (here, W2 answers typed on a phone, and all fall W1 answers until the engine is fixed), disclosure to test takers that their writing is scored by a machine, and a way to appeal.

The argument can go into the HTML report of the calibrated test with `mv_report()`, as one more section:

```
mv_report(fit, file = "writing-report.html",
          extra = list("Validity argument" = va))
```

18.5 Governance

An audit at launch is not enough. A testing program that uses AI components should also decide, and write down, the following. The *Standards* (American Educational Research Association et al. 2014) and the ITC/ATP guidelines (International Test Commission and Association of Test Publishers 2022) ask for most of it in general terms.

- **Ownership.** Who is accountable for each component, including one bought from a vendor, and who may change it.
- **Change control.** Every new engine version, prompt or training set is audited on a validity sample before it goes live, as Example 1 shows why.
- **Human oversight.** Which responses a person reads: A share chosen at random for monitoring, every response the engine is unsure about, and any group or item where the audit found a problem until it is fixed.
- **Disclosure and appeals.** Test takers and score users are told how scores were produced, and a score can be rescored by a person on request.
- **Monitoring.** The audit is rerun on each administration, with its results in the technical report.
- **Data.** What test-taker data a component sees, where it is processed, and how long it is kept, especially when a language model runs on a provider's servers (Chapter 17).

Guidance on AI in testing is moving quickly. The checks in this chapter rest on long-standing psychometric principles, but the regulations and professional guidelines that apply should be checked at the time of use.

18.6 Your turn

1. Rerun the audit of Example 1 with `mv_rules(subgroup_smd = 0.25)`. Which flags disappear, and what would the college then miss?
2. Audit only the spring window. Which problems does the audit find, and why is the fall engine version invisible?
3. Suppose the validity sample had included no phone answers to W2: Set their human scores to `NA` and rerun the audit. Does the differential scoring flag go away? What does that say about how the validity sample should be drawn?
4. Write the decision row of the validity argument for this college in your own words: Which responses would people read, what would the college tell students, and how would a student appeal?

Solutions

```
# 1. a looser rule
ai_audit(writing, human2 = "human2", group = "device", item = "prompt",
         period = "window", rules = mv_rules(subgroup_smd = 0.25))$checks
#> # A tibble: 6 x 4
#>   check                tested flagged status
#>   <chr>                 <int>   <int> <chr>
#> 1 human agreement           2       0 pass
#> 2 machine agreement         2       0 pass
#> 3 overall bias              2       0 pass
#> 4 subgroup agreement        4       0 pass
#> 5 differential scoring      2       0 pass
#> 6 drift                     2       0 pass

# 2. the spring window alone
ai_audit(writing[writing$window == "Spring", ], human2 = "human2", group =
 ↪  "device",
         item = "prompt")$checks
#> # A tibble: 5 x 4
#>   check                tested flagged status
#>   <chr>                 <int>   <int> <chr>
#> 1 human agreement           2       0 pass
#> 2 machine agreement         2       0 pass
#> 3 overall bias              2       0 pass
#> 4 subgroup agreement        4       0 pass
#> 5 differential scoring      2       1 block

# 3. without the phone answers to W2 in the validity sample
w3 <- writing
drop <- w3$device == "Phone" & w3$prompt == "W2"
w3$human1[drop] <- NA
w3$human2[drop] <- NA
ai_audit(w3, human2 = "human2", group = "device", item = "prompt")$differential
#> # A tibble: 2 x 6
#>   item  group reference     n      gap      se
#>   <chr> <chr> <chr>     <int>    <dbl>   <dbl>
#> 1 W1    Phone Laptop      217 -0.00220  0.0439
#> 2 W2    Phone Laptop        0 NA       NA
```

1. Both flags disappear: The phone gap (0.19 SD) and the fall drift (0.16 SD) are below 0.25. The college would place phone users lower than their writing warrants on W2, and every fall W1 test taker lower than a spring one with the same answer. A rule should be set from the size of a difference that matters for placement, before the data are seen.
2. The phone gap on W2 is found (in a smaller sample, so with a wider interval); drift

cannot be checked, because there is only one window. An audit can only find problems its data can show: Version changes need data from before and after.

3. The flag goes away, but not because the engine is fair: W2 has no phone answers with human scores left (`n` is 0), so there is no gap to compute, and the audit cannot flag what it cannot test. Its `checks` table shows one prompt tested instead of two. A validity sample must include every group and item the scores will be used for, in numbers large enough to detect a gap that matters.
4. For example: People read every W2 answer typed on a phone and every fall W1 answer until the engine is fixed, plus a random 10% of all answers and every answer near a placement cut. The placement letter says that writing is scored by an automated engine checked against trained raters, and that a student may ask for a human rescore within two weeks at no cost; the rescore replaces the machine score.

In the lab

> Lab 10, `run_lab("ai-audit")`, audits an AI-scored science test over two years before its scores are released: Agreement, fairness to English language learners, drift after an engine update, DIF on the test as scored, and a validity argument for the committee. The AI Hub collects the other AI tools in mirtverse.

Further reading

Kane (2013) sets out the argument-based approach, and the *Standards* (American Educational Research Association et al. 2014) the evidence a testing program owes. Williamson, Xi and Breyer (2012) give the framework for evaluating automated scoring that the agreement rules of mirtverse follow, and Johnson, Liu and McCaffrey (2022) show how to tell measurement bias from algorithmic bias with tools a psychometrician already knows. The ITC/ATP guidelines (2022) cover technology-based assessment more broadly.

Part II

Knowledge graphs

Chapter 19

Knowledge graphs and IRT

Learning goals. After this chapter you can

- describe what a test measures as a **knowledge graph** of concepts, prerequisites and items, and build one with `mv_kg_build()` or `as_mv_kg()`;
- check a graph for problems with `mv_kg_check()` and draw it with `gg_kg()`;
- read a graph's **Q-matrix** and list the mastery patterns its **attribute hierarchy** allows;
- turn a graph into a confirmatory multidimensional IRT model with `mv_kg_model()` and `mv_kg_irt()`, and test the graph's claims against simpler and stricter models;
- check a graph against data with `mv_kg_fit()` and explain item difficulty with it using `mv_kg_explain()`, with item metadata as extra predictors;
- draft a graph from item text with a language model using `mv_kg_extract()`, and review the draft against an expert graph with `mv_kg_compare()`.

Key terms: Knowledge graph, Q-matrix, attribute hierarchy, multidimensional IRT, likelihood-ratio test.

19.1 The ideas

Every test is built on claims about what it measures. A blueprint says which content each item covers; a curriculum says that some ideas must be learned before others; a teacher knows that an item about subtracting mixed numbers also needs borrowing. Most of these claims stay in documents and heads. A **knowledge graph** writes them down in a form a computer can check and a model can use. It has three kinds of parts.

- **Concepts** (also called skills, attributes or objectives): What the test is meant to tell us about.
- **Relations between concepts.** The most useful is **prerequisite of**: Equivalent fractions before adding fractions with unlike denominators. Prerequisite edges point forward in the learning order and should never form a cycle. A weaker relation, **related to**, links concepts where neither comes first.

- **Items**, each linked to the concepts it **measures**. These links are the rows of the **Q-matrix** (Tatsuoka 1984): One row per item, one column per concept, with a 1 where the item needs the concept.

The graph connects two traditions in psychometrics. In **multidimensional IRT**, the Q-matrix is the loading pattern of a confirmatory model: Each concept is a factor, and an item has a slope only on the concepts it measures (Chapter 4). In **cognitive diagnosis**, the same Q-matrix says which skills an item needs, and prerequisite edges form an **attribute hierarchy** (Leighton et al. 2004; Templin and Bradshaw 2014): A student can master a concept only after mastering its prerequisites, so some mastery patterns are impossible. Knowledge space theory (Doignon and Falmagne 1999) builds a whole theory of assessment on such structures.

A graph is a set of claims, and each claim can be wrong. An item can be linked to the wrong concept, a prerequisite can be missing or reversed, and two concepts can be impossible to tell apart because the same items measure them. The work in this chapter is to make the claims explicit, check what can be checked without data, and then let the data speak.

19.2 Example 1: A graph for fraction subtraction

Tatsuoka's fraction subtraction data are the best-known real data in cognitive diagnosis: 536 middle school students answered 20 items such as $4\frac{1}{3} - 2\frac{4}{3}$, and the items were coded on eight attributes (Tatsuoka 2002; DeCarlo 2011; Torre and Douglas 2004). mirtverse stores the responses as `fraction_subtraction` and the Q-matrix as a knowledge graph, `fraction_subtraction_kg`.

```
resp <- fraction_subtraction
kg <- fraction_subtraction_kg
```

```
kg
#> <mv_kg> Fraction subtraction (Tatsuoka): 8 concept(s), 20 item(s)
#> Edges: 0 prerequisite, 0 related, 56 measures
#> Item metadata: text
tidy(kg)
#> # A tibble: 28 x 6
#>   node               kind    label                 description links depth
#>   <chr>              <chr>   <chr>                 <chr>       <dbl> <dbl>
#> 1 convert_whole      concept Convert a whole numb~ Attribute ~     3     0
#> 2 separate_whole     concept Separate a whole num~ Attribute ~    13     0
#> 3 simplify_first     concept Simplify before subt~ Attribute ~     3     0
#> 4 common_denominator concept Find a common denomi~ Attribute ~     5     0
#> 5 borrow_whole       concept Borrow from the whol~ Attribute ~     8     0
#> 6 column_borrow      concept Column borrow to sub~ Attribute ~     2     0
#> # i 22 more rows
```

The graph has eight concepts and 20 items, with 56 item-to-concept links. No prerequisite

edges were published with the Q-matrix, so the graph has none yet. `tidy()` counts the items linked to each concept (`links`). `subtract_numerators` is needed by 19 of the 20 items, while `column_borrow` is needed by only two. Before any data are used, `mv_kg_check()` looks for problems in the structure itself.

```
mv_kg_check(kg)
#> <mv_kg_check> 0 block, 1 review, 0 info
#> * [review] few_items: column_borrow is measured by 2 item(s); at
#>   least 3 are needed to estimate it as its own dimension.
```

The check finds one thing to review: Two items are too few to estimate `column_borrow` as its own dimension. A concept that nearly every item needs, like `subtract_numerators`, has the opposite problem: It is hard to separate from overall proficiency. Neither is an error in the graph, but both limit what a model can learn about these concepts.

```
gg_kg(kg)
```

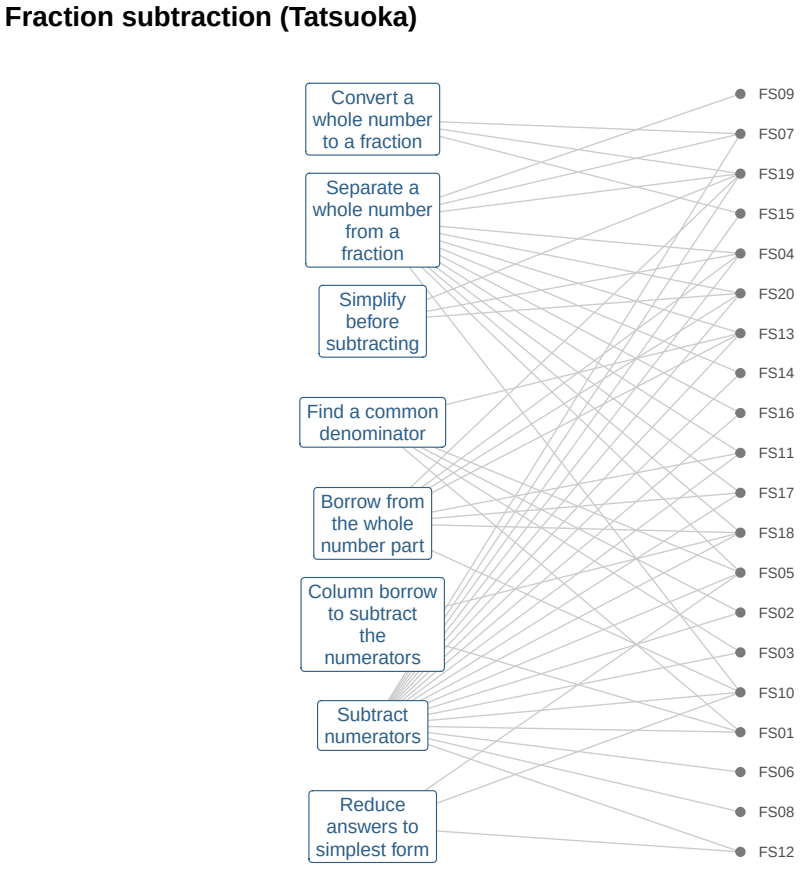


Figure 19.1: The fraction subtraction graph: Eight attributes on the left, each joined to the items that need it.

Now suppose a mathematics teacher states two prerequisites as hypotheses: Borrowing from the whole number needs the whole number to be separated from the fraction first, and column borrowing needs subtracting numerators. We add these edges to the same concepts and items.

```
concepts <- tidy(kg)
concepts <- concepts[concepts$kind == "concept", ]
kg_h <- mv_kg_build(
  concepts = data.frame(concept = concepts$node, label = concepts$label),
  edges = data.frame(from = c("separate_whole", "subtract_numerators"),
                     to = c("borrow_whole", "column_borrow")),
  items = mv_kg_qmatrix(kg),
  name = "Fraction subtraction with two prerequisites")
kg_h
#> <mv_kg> Fraction subtraction with two prerequisites: 8 concept(s), 20 item(s)
#> Edges: 2 prerequisite, 0 related, 56 measures
#> Learning order (by prerequisite depth):
#>   0: convert_whole, separate_whole, simplify_first, common_denominator,
 ↪  subtract_numerators, reduce_answer
#>   1: borrow_whole, column_borrow
```

The print-out now gives a learning order. The hierarchy has a practical consequence: With eight concepts there are $2^8 = 256$ possible mastery patterns, and each prerequisite rules out the patterns where a concept is mastered without its prerequisite.

```
patterns <- mv_kg_hierarchy(kg_h)
nrow(patterns)
#> [1] 144
attr(patterns, "possible")
#> [1] 256
```

Two edges remove 112 of the 256 patterns. A diagnostic model that respects the hierarchy has fewer classes to estimate, and the hierarchy itself can be tested by comparing models with and without it.

19.3 Example 2: From graph to IRT model

`mv_kg_model()` writes the confirmatory mirt model a graph implies, and `mv_kg_irt()` fits it. Eight correlated dimensions on 536 students is a large model that takes many minutes to fit, so we work with a smaller graph: The three concepts with the clearest roles, separating the whole number, borrowing, and finding a common denominator. `mv_kg_subset()` keeps these concepts, the prerequisite between two of them, and the 16 items that need at least one of them.

```
kg3 <- mv_kg_subset(kg_h, c("separate_whole", "borrow_whole",
 ↪  "common_denominator"))
```

```
kg3
#> <mv_kg> Fraction subtraction with two prerequisites: 3 concept(s), 16 item(s)
#> Edges: 1 prerequisite, 0 related, 26 measures
#> Learning order (by prerequisite depth):
#>   0: separate_whole, common_denominator
#>   1: borrow_whole
q <- mv_kg_qmatrix(kg3)
resp3 <- resp[rownames(q)]
spec <- mv_kg_model(kg3)
cat(attr(spec, "item_syntax"))
#> separate_whole = FS04, FS05, FS07, FS09, FS10, FS11, FS13, FS14, FS16, FS17,
 ↪  FS18, FS19, FS20
#> common_denominator = FS01, FS02, FS03, FS05, FS13
#> borrow_whole = FS04, FS10, FS11, FS13, FS17, FS18, FS19, FS20
#> COV = separate_whole*common_denominator, separate_whole*borrow_whole,
 ↪  common_denominator*borrow_whole
```

Each concept is a factor, and each item loads on the concepts its row of the Q-matrix names; items that need two concepts load on both. By default all concepts correlate. The `structure` argument changes that: `structure = "graph"` lets only concepts joined by an edge correlate, a strict reading of the graph in which concepts without an edge are unrelated.

```
m_uni <- mirt(resp3, 1, verbose = FALSE)
m_graph <- mv_kg_irt(kg3, resp3)
m_strict <- mv_kg_irt(kg3, resp3, structure = "graph")
anova(m_uni, m_graph)
#>          AIC SABIC   HQ  BIC logLik     X2 df p
#> m_uni   7739  7775 7793 7876  -3838
#> m_graph 7371  7421 7447 7564  -3641 394.06 13 0
anova(m_strict, m_graph)
#>           AIC SABIC   HQ  BIC logLik      X2 df p
#> m_strict 7641  7689 7713 7825  -3778
#> m_graph  7371  7421 7447 7564  -3641 273.834  2 0
```

Two comparisons answer two questions. Against a single dimension, the three-concept model fits far better: The likelihood-ratio statistic is about 394 on 13 degrees of freedom, and BIC drops by more than 300. The items measure more than one proficiency, in the way the graph describes. Against the strict model, the correlated model also wins clearly: Finding a common denominator has no edge in the graph, yet it correlates with the other two concepts.

```
round(summary(m_graph, verbose = FALSE)$fcor, 2)
#>                   separate_whole common_denominator borrow_whole
#> separate_whole              1.00              0.77        0.38
#> common_denominator          0.77              1.00        0.39
```

```
#> borrow_whole                0.38              0.39         1.00
```

Separating the whole number and finding a common denominator correlate at about 0.77, much more than the two concepts the teacher linked (about 0.37). This is a typical finding, not a failure: A prerequisite edge says something about the *order* of learning, and concepts that are learned in the same lessons correlate whether or not one is a prerequisite of the other. The correlations are a reason to look again at the graph, for example to ask whether a related-to edge is missing.

The fitted model is an ordinary mirt model, so everything from the earlier chapters works on it, and the graph travels with it.

```
pars <- tidy(m_graph)
head(pars[pars$estimate != 0, ])
#> # A tibble: 6 x 5
#>   item  itemtype term  estimate std.error
#>   <chr> <chr>    <chr>    <dbl>     <dbl>
#> 1 FS01  2PL      a2       5.13         NA
#> 2 FS01  2PL      d        1.08         NA
#> 3 FS02  2PL      a2       7.39         NA
#> 4 FS02  2PL      d        2.31         NA
#> 5 FS03  2PL      a2      13.8          NA
#> 6 FS03  2PL      d        0.943        NA
attr(m_graph, "mv_kg")
#> <mv_kg> Fraction subtraction with two prerequisites: 3 concept(s), 16 item(s)
#> Edges: 1 prerequisite, 0 related, 26 measures
#> Learning order (by prerequisite depth):
#>   0: separate_whole, common_denominator
#>   1: borrow_whole
```

`as_mv_kg()` also goes the other way: Given any confirmatory mirt model, it reads the loading pattern back into a graph, which is a quick way to draw a model you already have.

19.4 Example 3: Checking the graph against data

The comparisons above test the graph as a whole. `mv_kg_fit()` looks for the places where the data disagree with it. For each item, it asks whether the item agrees more with another concept than with its own: It correlates the item with the rest score of every concept, corrected for how reliable each rest score is. For each pair of concepts, it turns the factor correlations of the correlated model into **partial correlations**, the correlation of two concepts once the others are held constant. Concepts that are directly related keep a clear partial correlation; concepts related only through others do not. We can pass in the model we already have.

```
check <- mv_kg_fit(kg3, resp3, model = m_graph)
check
#> <mv_kg_fit> Fraction subtraction with two prerequisites: 3 concept(s), 16
 ↪  item(s), N = 536
#> 2 review, 0 info
#> * [review] weak_link: FS13 is linked to separate_whole,
#>   common_denominator, borrow_whole but agrees much less with
#>   common_denominator than with separate_whole (r = 0.57 vs 0.70);
#>   drop the link to common_denominator?
#> * [review] missing_relation: separate_whole and
#>   common_denominator have no edge but a partial correlation of
#>   0.73, and their items are about equally hard; add a related
#>   edge?
#> Item links shape the concept correlations: Fix item flags first
#> and refit before acting on relation flags.
tidy(check, "relations")
#> # A tibble: 3 x 10
#>   concept_a  concept_b edge  from  to    type      r partial ease_a ease_b
#>   <chr>      <chr>     <chr> <chr> <chr> <chr> <dbl>   <dbl>  <dbl>  <dbl>
#> 1 separate_~ common_d~ none  <NA>  <NA>  <NA>  0.771   0.731  0.487  0.500
#> 2 separate_~ borrow_w~ sepa~ sepa~ borr~ prer~ 0.382   0.137  0.487  0.410
#> 3 common_de~ borrow_w~ none  <NA>  <NA>  <NA>  0.391   0.164  0.500  0.410
```

Every item agrees best with a concept it is linked to, so there is no `wrong_link` flag. One item gets a `weak_link` flag instead: FS13 (3 3/8 - 2 5/6) is linked to three concepts but agrees much less with finding a common denominator than with separating the whole number. Look at the item before acting on that: Eighths and sixths do need a common denominator, so the link is right on content. Nearly every item with a whole number part also needs separating and borrowing, which share most of their items, so those concepts' rest scores track FS13 more closely. This is a flag to reject, and a reminder that the checks are screening heuristics.

The relation evidence makes the earlier observation precise. Holding borrowing constant, separating the whole number and finding a common denominator still have a partial correlation of about 0.73, while the teacher's prerequisite (separate_whole -> borrow_whole) drops to about 0.14. Because the two concepts' items are about equally hard, the suggestion is a related edge rather than a prerequisite.

```
gg_kg_fit(check, items = "all")
```

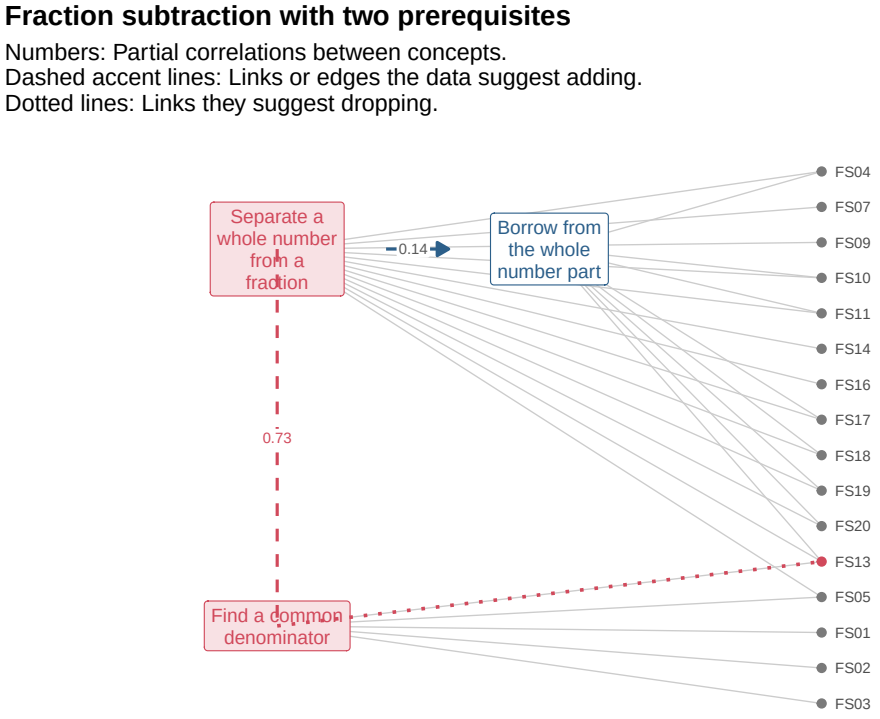


Figure 19.2: The evidence on the three-concept graph: Partial correlations next to each pair, the suggested related edge as a dashed line, and the weak link of FS13 as a dotted line.

`mv_kg_refine(check, "relations")` would add the edge and leave the item links alone. Whether to do so is a content question: Both skills come up when students handle mixed numbers with unlike denominators, so a related edge is plausible, but the data alone cannot say why the concepts go together.

A graph also makes claims about difficulty. `mv_kg_explain()` fits an explanatory Rasch model with the eirm package (Bulut et al. 2021) in which the graph predicts each item's difficulty: One difficulty per concept (a linear logistic test model, with a random item effect for what the concepts leave unexplained), or the item's depth in the learning order.

```
explain <- mv_kg_explain(kg, resp)
explain
#> <mv_kg_explain> Fraction subtraction (Tatsuoka): Item difficulty explained by
 ↪  concepts
#> 20 items, N = 536. The graph explains 88% of the item variance.
#>                label difficulty   se p.value
#>        convert_whole       2.10 0.29  <0.001
#>       separate_whole      -0.18 0.23    0.42
#>       simplify_first      -0.40 0.33    0.22
#>   common_denominator       1.32 0.24  <0.001
#>         borrow_whole       2.33 0.29  <0.001
```

```
#>        column_borrow      -0.01 0.33    0.98
#>  subtract_numerators      -1.69 0.22  <0.001
#>        reduce_answer       0.11 0.28    0.70
#> Items the graph predicts poorly: FS09 (-0.91).
```

The eight attributes explain most of the differences in item difficulty. Borrowing from the whole number and converting a whole number to a fraction add the most difficulty, and subtracting numerators, which almost every item needs, the least. The teacher's two prerequisites, in contrast, say little about difficulty:

```
mv_kg_explain(kg_h, resp, predictors = "depth")
#> <mv_kg_explain> Fraction subtraction with two prerequisites: Item difficulty
 ↪  explained by learning-order depth
#> 20 items, N = 536. The graph explains 64% of the item variance.
#>                 label difficulty   se p.value
#>  Learning-order depth       0.47 0.49  0.3354
#>    Number of concepts       0.63 0.21  0.0024
```

Items that need more attributes are harder, but once that is taken into account, an item's depth in the teacher's learning order adds little. A prerequisite that does not show up as a difference in difficulty is not necessarily wrong, but it is a claim the data do not support yet.

Other things known about the items can join the concepts. A graph carries **item metadata**: `fraction_subtraction_kg` has the text of each problem, and `mv_kg_annotate()` adds more columns. Teachers often say that larger denominators make a problem harder. Take the largest denominator in each problem from its text and add it to the model:

```
denominator <- vapply(regmatches(kg$meta$text, gregexpr("/[0-9]+", kg$meta$text)),
                      function(d) max(as.numeric(sub("/", "", d))), numeric(1))
kg_m <- mv_kg_annotate(kg, data.frame(item = kg$meta$item, denominator =
 ↪  denominator))
tidy(kg_m, "items")[1:5, ]
#> # A tibble: 5 x 5
#>   item  concepts                              n_concepts text  denominator
#>   <chr> <chr>                                      <int> <chr>       <dbl>
#> 1 FS01  common_denominator, column_borrow, s~          3 5/3 ~           4
#> 2 FS02  common_denominator, subtract_numerat~          2 3/4 ~           8
#> 3 FS03  common_denominator, subtract_numerat~          2 5/6 ~           9
#> 4 FS04  separate_whole, simplify_first, borr~          4 3 1/~           2
#> 5 FS05  separate_whole, common_denominator, ~          4 4 3/~          10
mv_kg_explain(kg_m, resp, covariates = "denominator")
#> <mv_kg_explain> Fraction subtraction (Tatsuoka): Item difficulty explained by
 ↪  concepts and item metadata
#> 20 items, N = 536. The graph explains 88% of the item variance.
#>                  label difficulty   se p.value
```

```
#>        convert_whole       2.09 0.30  <0.001
#>       separate_whole      -0.17 0.24    0.47
#>       simplify_first      -0.44 0.38    0.24
#>   common_denominator       1.34 0.26  <0.001
#>         borrow_whole       2.34 0.29  <0.001
#>        column_borrow       0.00 0.34    1.00
#>  subtract_numerators      -1.70 0.24  <0.001
#>        reduce_answer       0.15 0.35    0.66
#>          denominator      -0.01 0.05    0.83
#> Items the graph predicts poorly: FS09 (-0.90).
```

Once the attributes are in the model, the size of the denominator adds nothing: Its effect is close to zero and the share of item variance explained stays at 88%. What makes these items hard is the steps they need, not the numbers in them. The same check works for any item feature, such as word count, format, or whether a person or a language model wrote the item.

19.5 Example 4: Drafting a graph from item text

Writing a Q-matrix is slow, expert work: Someone has to read every item and decide which skills it needs. A language model can read items too, so it can draft the graph, and the expert's job becomes reviewing the draft. `mv_kg_extract()` sends the item text to a model (Chapter 17) and asks for the item links and the prerequisites between concepts. Give it the concepts, as here, and it only links items to them; leave them out and it proposes the concepts too. The function is **experimental**.

We run it from the example recordings that come with mirtverse, so no model or API key is needed. Those responses were written by the package authors, with mistakes planted for the checks to catch; they are not the output of a live model. With `mv_ask_setup()` the same call goes to Claude, an OpenAI model or a local model.

```
old <- mv_llm_replay()
draft <- mv_kg_extract(kg, subject = "Middle school fraction subtraction")
draft
#> <mv_kg_draft> (EXPERIMENTAL) Drafted by recorded responses from the concepts
 ↪  given
#> <mv_kg> Fraction subtraction (Tatsuoka) (drafted): 8 concept(s), 20 item(s)
#> Edges: 7 prerequisite, 0 related, 54 measures
#> Item metadata: text
#> Learning order (by prerequisite depth):
#>   0: convert_whole, separate_whole, simplify_first, subtract_numerators
#>   1: common_denominator, borrow_whole, reduce_answer
#>   2: column_borrow
#> Checks on the model's answer: 5 flag(s)
#> * [review] unknown_item: The model linked FS21, which is not one
```

```
#>   of the items; the link was dropped.
#> * [review] unknown_concept: FS13 was linked to
#>   find_common_denominator, which is not a concept; the link was
#>   dropped.
#> * [review] two_way: The model gave the prerequisite both ways
#>   between simplify_first and reduce_answer; simplify_first ->
#>   reduce_answer was kept (higher confidence).
#> * [review] cycle: The prerequisites formed a cycle through
#>   subtract_numerators, borrow_whole, column_borrow; column_borrow
#>   -> subtract_numerators (confidence 0.3) was removed.
#> * [info] new_concept: The model proposed a new concept,
#>   subtract_whole_numbers; it was left out because the concepts
#>   were given.
#> Review the draft with mv_kg_check() and check it against data with mv_kg_fit().
```

Before the graph is built, the model's answer is checked. Some problems are simply errors and are fixed on the spot: A link to an item FS21 that does not exist, and a concept the model was told not to add. Others are fixed in a stated way: Where the model gave a prerequisite both ways, the direction with higher confidence is kept, and the edge that closed a cycle (the one with the lowest confidence) is removed. One fix costs information: FS13 was linked to `find_common_denominator`, a misspelling, so that link was dropped. Every model link comes with a reason:

```
reasons <- tidy(draft, "reasons")
reasons[reasons$from %in% c("FS04", "FS08"), c("from", "to", "reason")]
#> # A tibble: 5 x 3
#>   from  to                 reason
#>   <chr> <chr>              <chr>
#> 1 FS04  separate_whole     Work with the whole numbers and fractions sep~
#> 2 FS04  simplify_first     3/2 is improper and can be simplified to 1 1/~
#> 3 FS04  subtract_numerators Subtract the fractional parts.
#> 4 FS08  subtract_numerators Like denominators; subtract the numerators.
#> 5 FS08  reduce_answer      The answer 0/3 should be written as 0.
```

Now compare the draft with the expert Q-matrix:

```
mv_kg_compare(draft, kg)
#> <mv_kg_compare> Fraction subtraction (Tatsuoka) (drafted) vs Fraction
 ↪  subtraction (Tatsuoka)
#> Item links: 53 of the 56 reference links are in the graph (recall
#>   0.95), and 53 of the graph's 54 links are in the reference
#>   (precision 0.98).
#> 16 of 20 items are linked to exactly the same concepts.
#> Items that differ: FS01 (missing column_borrow), FS04 (missing
```

```
#>   borrow_whole), FS08 (extra reduce_answer), FS13 (missing
#>   common_denominator).
#> Prerequisites: The graph has 7; the reference has none to compare with.
options(old)
```

The model and the expert agree on 16 of the 20 items, and on 53 of the 56 expert links. The four differences are the places to look, and they are not all the model's mistakes:

- **FS13** is the misspelled link. The checks caught it, and the expert is right: Eighths and sixths need a common denominator.
- **FS01** (5/3 - 3/4): The expert codes column borrowing and the model does not. This attribute is the hardest to apply from the text alone, and the coding is the expert's call.
- **FS04** (3 1/2 - 2 3/2): The model leaves out borrowing. Its reason is that 2 3/2 can be simplified to 3 1/2 first, after which there is nothing to borrow; the expert codes the usual procedure, which borrows. Both readings are defensible, and they describe different solution strategies.
- **FS08** (2/3 - 2/3): The model adds reducing the answer, because 0/3 has to be written as 0. Again defensible, and a question for the item writers: Is that step part of what the item is meant to measure?

The draft's prerequisites have nothing to compare with, because none were published with the Q-matrix. They are hypotheses, and Example 3 showed how to check such claims with data: Partial correlations with `mv_kg_fit()` and depth with `mv_kg_explain()`.

The lesson is the one that runs through this part of the book: A model's output is a draft to be checked. Here three kinds of checks work together. The **answer checks** catch errors in what the model returned, the **comparison** with an expert graph shows where two readings of the items differ, and the **data** (Example 3) say which links and edges the students' answers support. A draft that agrees with the expert on most items, with its disagreements listed, is a useful second opinion; it is not a replacement for the expert.

19.6 Your turn

1. Add a third hypothesis to `kg_h`: Converting a whole number to a fraction (`convert_whole`) is a prerequisite of borrowing (`borrow_whole`). How many mastery patterns are left?
2. Add a cycle by also making `borrow_whole` a prerequisite of `separate_whole`. What does `mv_kg_check()` report, and what happens to `mv_kg_hierarchy()`?
3. Fit the three-concept model with `structure = "orthogonal"`. How does it compare with the correlated model, and what does that say about scoring the three concepts separately?
4. Draw `kg3` with `gg_kg()`, highlighting `common_denominator`. Which items does it share with `separate_whole`?

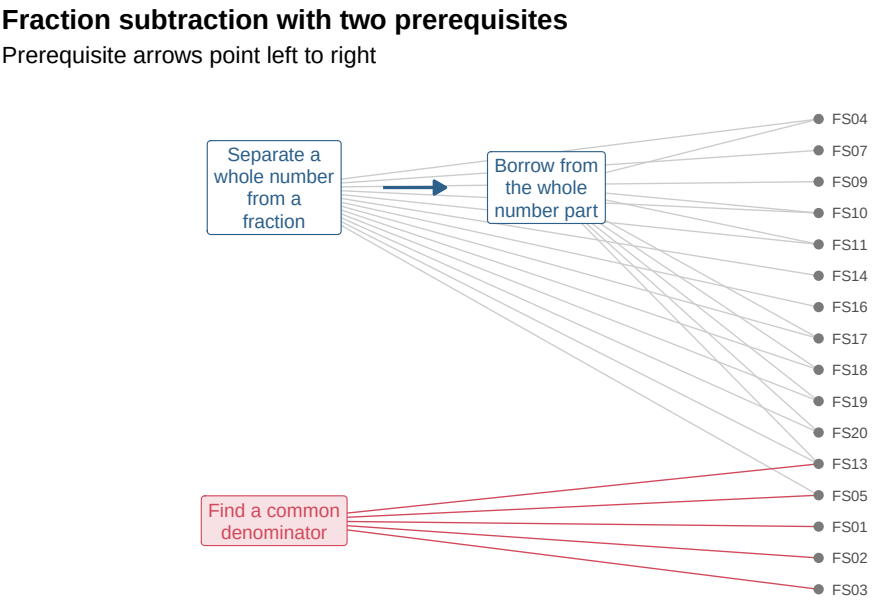
Solutions

```
# 1. a third prerequisite
edges3 <- data.frame(from = c("separate_whole", "subtract_numerators",
 ↪  "convert_whole"),
                     to = c("borrow_whole", "column_borrow", "borrow_whole"))
kg_h3 <- mv_kg_build(data.frame(concept = concepts$node, label = concepts$label),
                     edges = edges3, items = mv_kg_qmatrix(kg))
nrow(mv_kg_hierarchy(kg_h3))
#> [1] 120

# 2. a cycle
kg_cyc <- mv_kg_build(data.frame(concept = concepts$node),
                      edges = rbind(edges3, data.frame(from = "borrow_whole",
                                                       to = "separate_whole")),
                      items = mv_kg_qmatrix(kg))
mv_kg_check(kg_cyc)
#> <mv_kg_check> 1 block, 1 review, 3 info
#> * [block] cycle: Prerequisite edges form a cycle through
#>   separate_whole, borrow_whole, so no learning order exists.
#> * [review] few_items: column_borrow is measured by 2 item(s); at
#>   least 3 are needed to estimate it as its own dimension.
#> * [info] isolated: simplify_first has no relations to other
#>   concepts.
#> * [info] isolated: common_denominator has no relations to other
#>   concepts.
#> * [info] isolated: reduce_answer has no relations to other
#>   concepts.
try(mv_kg_hierarchy(kg_cyc))
#> Error : The prerequisite edges contain a cycle; see mv_kg_check().

# 3. uncorrelated concepts
m_orth <- mv_kg_irt(kg3, resp3, structure = "orthogonal")
anova(m_orth, m_graph)
#>          AIC SABIC   HQ  BIC logLik      X2 df p
#> m_orth  7642  7689 7712 7822  -3779
#> m_graph 7371  7421 7447 7564  -3641 276.773  3 0

# 4. shared items
gg_kg(kg3, highlight = "common_denominator")
```




```
q[q[, "common_denominator"] == 1 & q[, "separate_whole"] == 1, ]
#>      separate_whole common_denominator borrow_whole
#> FS05              1                 1            0
#> FS13              1                 1            1
```

1. Borrowing now needs both separating and converting, so more patterns are ruled out: 120 of the 256 remain.
2. The check reports a `cycle` block flag: Separating the whole number would have to come both before and after borrowing, so no learning order exists. `mv_kg_hierarchy()` stops with an error, because a cycle allows no consistent set of patterns.
3. The orthogonal model fits much worse than the correlated one (the likelihood-ratio statistic is about 277 on 3 degrees of freedom). The concepts share a lot of variance, so three separate scores would repeat much of the same information; reporting them separately needs care (Chapter 12).
4. Two items, FS05 and FS13, need both concepts; FS13 also needs borrowing. The other common denominator items (FS01 to FS03) involve no whole numbers. Items that load on two concepts help the model separate the concepts from their correlation.

In the lab

Lab 11, Repairing a knowledge graph (`run_lab("kg-repair")`): A district's curriculum team has written a knowledge graph for its Grade 5 fractions test, and diagnostic reports will follow it. Check the graph against the students' answers with `mv_kg_fit()`, find an item linked to the wrong concept and a missing prerequisite, repair the graph with `mv_kg_refine()`, and show what the repair changes.

Lab 12, Reviewing an AI-drafted graph (`run_lab("kg-draft")`): The same district asks a language model to draft the graph from the item text. Review the draft: Read the checks on the model's answer, compare it with the curriculum team's graph, test both against the students' answers, and decide what to keep.

Further reading

Tatsuoka (1984) introduced the Q-matrix with the fraction subtraction data; DeCarlo (2011) and de la Torre and Douglas (2004) reanalysed them with diagnostic models. Leighton, Gierl and Hunka (2004) describe the attribute hierarchy method, and Templin and Bradshaw (2014) show how to test hierarchies with diagnostic classification models. Doignon and Falmagne (1999) develop knowledge space theory, the mathematical theory of prerequisite structures in assessment.

Chapter 20

Knowledge tracing

Learning goals. After this chapter you can

- explain how **Bayesian knowledge tracing** (BKT) follows a student's mastery of a skill from one practice attempt to the next, and what its four parameters mean;
- fit BKT to practice logs with `mv_kt_fit()`, including the variants with forgetting and with a guess and a slip per item, and judge the models by cross-validation with `mv_kt_cv()`;
- compare BKT with a Rasch model on the same logs with `mv_kt_irt()`, and relate BKT's guess parameters to IRT item difficulty;
- draw mastery over time and on a knowledge graph with `gg_kt()`, and check a graph's prerequisites against practice logs with `mv_kt_prereq()`.

Key terms: Knowledge tracing, knowledge graph, attribute hierarchy, cross-validation, likelihood-ratio test.

The functions in this chapter are experimental: Their arguments and output may still change.

20.1 The ideas

IRT assumes that a student's ability stays the same while they answer the test. That is reasonable for a one-hour exam, but not for an online tutor, where students practise a skill until they get it. In practice logs, the interesting question is not how able a student is, but whether they have learned the skill yet, and when.

Bayesian knowledge tracing (Corbett and Anderson 1995) answers it with a hidden Markov model. Each skill is either **known** or **not known**. Four probabilities describe each skill:

- `prior`: The probability that a student knows the skill before the first attempt;
- `learn`: The probability that a student who does not know the skill learns it after an attempt (with feedback);
- `guess`: The probability of a correct answer without knowing the skill;

- `slip`: The probability of a wrong answer while knowing it.

After every answer, Bayes' rule updates the probability that the student knows the skill. A correct answer raises it, a wrong one lowers it, and then the chance of learning moves it up before the next attempt:

$$P(L_t \mid \text{correct}) = \frac{P(L_t)\,(1-S)}{P(L_t)\,(1-S) + [1-P(L_t)]\,G},$$

$$P(L_t \mid \text{wrong}) = \frac{P(L_t)\,S}{P(L_t)\,S + [1-P(L_t)]\,(1-G)},$$

$$P(L_{t+1}) = P(L_t \mid \text{answer}) + [1-P(L_t \mid \text{answer})]\,T,$$

where S is the slip, G the guess and T the learn rate. The parameters are estimated by expectation-maximization, skill by skill, as the pyBKT library does (Badrinath et al. 2021). Bulut et al. (2023) introduce BKT with pyBKT for psychometricians; this chapter follows their examples in R.

BKT and IRT are close relatives (Deonovic et al. 2018). The guess parameter of a skill, or of an item when each item has its own (the item difficulty effect model, KT-IDEM; Pardos and Heffernan (2011)), plays the role of item difficulty: Items that are easy to answer without knowing the skill are easy items. What BKT adds is learning: The latent state changes during the logs.

20.2 Example 1: Plotting skills in a Cognitive Tutor

```
logs <- cognitive_tutor
```

`cognitive_tutor` holds the practice logs used in the pyBKT examples (Stamper et al. 2010): 587 students' first attempts at 16,857 problem steps in a mathematics tutor, each tagged with one of 12 skills. Following Bulut et al. (2023), we start with the six skills about plotting numbers on a number line.

```
head(logs)
#> # A tibble: 6 x 7
#>   student problem    step    skill correct opportunity time
#>   <chr>   <chr>      <chr>   <chr>   <int>       <int> <dttm>
#> 1 745Yh   RATIO2-001 Simpli~ Calc~       1           1 2006-11-14 10:18:05
#> 2 745Yh   RATIO2-001 Simpli~ Calc~       0           2 2006-11-14 10:18:17
#> 3 745Yh   RATIO2-012 Simpli~ Calc~       0           3 2006-11-14 10:50:57
#> 4 745Yh   RATIO2-012 Simpli~ Calc~       1           4 2006-11-14 10:51:14
#> 5 745Yh   RATIO2-054 Simpli~ Calc~       0           5 2006-11-28 09:53:47
#> 6 745Yh   RATIO2-054 Simpli~ Calc~       1           6 2006-11-28 09:54:14
plot_skills <- grep("^Plot", unique(logs$skill), value = TRUE)
```

```
kt <- mv_kt_fit(logs, skills = plot_skills, seed = 1)
kt
#> <mv_kt> (EXPERIMENTAL) Bayesian knowledge tracing
#> 6 skill(s), 258 student(s), 4336 attempt(s)
#>                                    skill prior learn guess  slip
#>                        Plot whole number 0.930 0.449 0.500 0.071
#>        Plot terminating proper fraction 0.574 0.027 0.318 0.328
#>                   Plot imperfect radical 0.282 0.134 0.080 0.289
#>  Plot non-terminating improper fraction 0.728 0.231 0.000 0.309
#>                                 Plot pi 0.676 0.624 0.051 0.108
#>              Plot decimal - thousandths 0.139 0.397 0.203 0.338
#> Fit on the same logs: RMSE 0.450, accuracy 0.674, AUC 0.745
```

The parameters tell a story about each skill. Plotting whole numbers is known by almost everyone from the start (`prior` 0.93), so the model has little to learn from these logs; its guess reaches the upper bound of 0.5 that keeps known and not-known students apart. Plotting pi is learned quickly (`learn` 0.62), and plotting terminating proper fractions slowly (0.03), with a high slip: Students who probably know it still make many mistakes. A skill with a guess of 0 and a slip of 0.31, such as plotting non-terminating improper fractions, says that a correct answer is strong evidence of knowing the skill but a wrong one is weak evidence of not knowing it.

```
gg_kt(kt, max_attempts = 15)
```

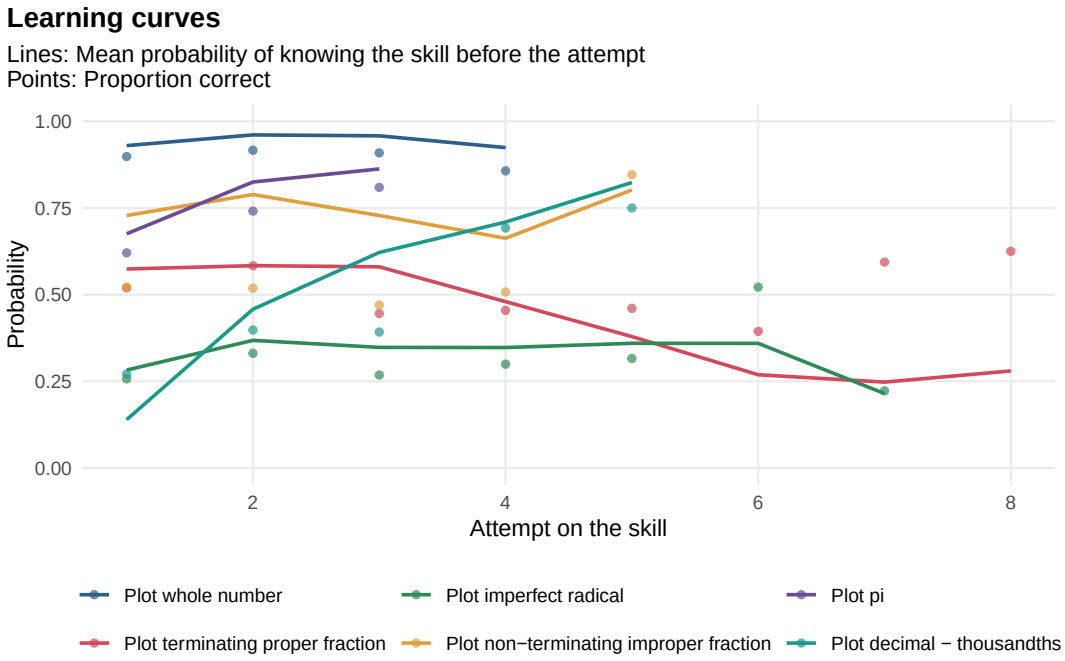


Figure 20.1: Learning curves for the plotting skills: The mean probability of knowing each skill before each attempt (lines) and the proportion of correct answers (points).

One student's record shows the updating at work.

```
student <- names(which.max(table(logs$student[logs$skill %in% plot_skills])))
gg_kt(kt, student = student)
```

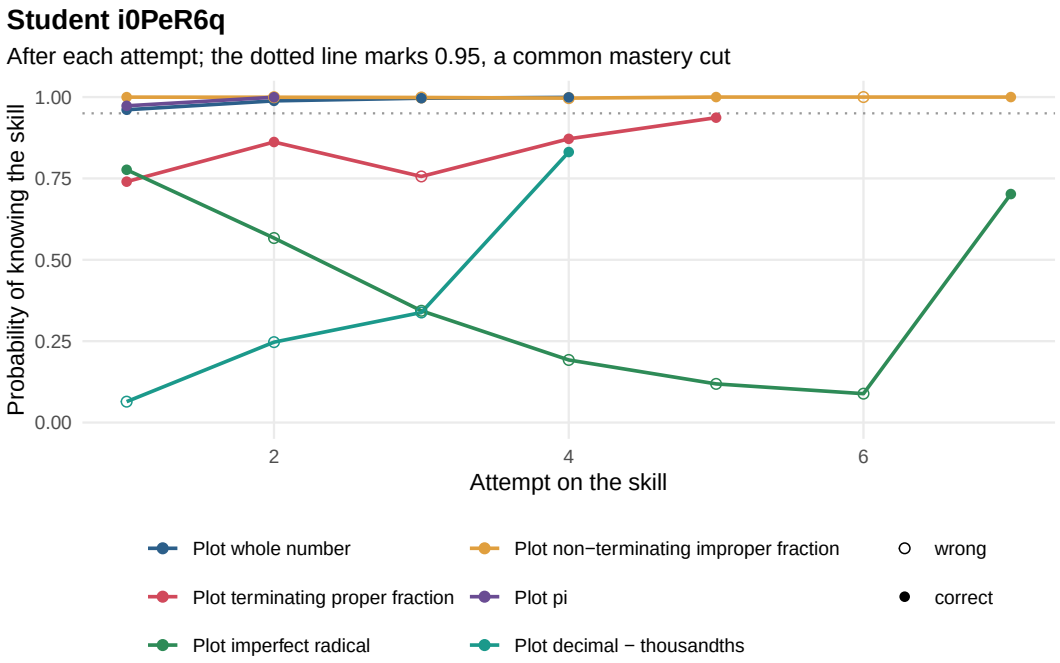


Figure 20.2: One student's probability of knowing each plotting skill after every attempt: Filled points are correct answers.

20.2.1 Variants

pyBKT's two most used variants are available as arguments: `forgets = TRUE` lets a known skill be forgotten, and `multigs = TRUE` gives each item (here each problem) its own guess and slip.

```
kt_f <- mv_kt_fit(logs, skills = plot_skills, forgets = TRUE, seed = 1)
kt_g <- mv_kt_fit(logs, item = "problem", skills = plot_skills, multigs = TRUE,
↪  seed = 1)
rbind(standard = glance(kt), forgets = glance(kt_f), multigs = glance(kt_g))
#> # A tibble: 3 x 9
#>   n_skills n_students n_attempts loglik   AIC   BIC  rmse accuracy   auc
#> *    <int>      <int>      <int>  <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>
#> 1        6        258       4336 -2552. 5163. 5354. 0.450    0.674 0.745
#> 2        6        258       4336 -2551. 5174. 5404. 0.450    0.674 0.745
#> 3        6        258       4336 -2313. 5054. 6418. 0.424    0.726 0.805
```

Forgetting adds nothing here: The log-likelihood barely moves and AIC goes up. Item guess and slip fit much better (AIC drops by more than 100 and the AUC rises from 0.75 to 0.81), but they add 184 parameters, and BIC prefers the standard model. Fit on the same logs always flatters the bigger model, so the fair test is new students.

```
cv <- mv_kt_cv(logs, skills = plot_skills, folds = 3, seed = 1)
cv_g <- mv_kt_cv(logs, item = "problem", skills = plot_skills, multigs = TRUE,
```

```
                folds = 3, seed = 1)
merge(cv[cv$fold == "mean", c("skill", "rmse", "auc")],
      cv_g[cv_g$fold == "mean", c("skill", "rmse", "auc")],
      by = "skill", suffixes = c("_standard", "_multigs"))
#>                                   skill rmse_standard auc_standard
#> 1             Plot decimal - thousandths         0.472        0.622
#> 2                 Plot imperfect radical         0.437        0.643
#> 3 Plot non-terminating improper fraction         0.482        0.651
#> 4                               Plot pi         0.462        0.581
#> 5       Plot terminating proper fraction         0.498        0.569
#> 6                     Plot whole number         0.292        0.514
#>   rmse_multigs auc_multigs
#> 1        0.454       0.705
#> 2        0.440       0.657
#> 3        0.492       0.629
#> 4        0.453       0.667
#> 5        0.471       0.704
#> 6        0.284       0.696
```

On held-out students the item model still predicts better for most skills, especially the ones with many problems, such as plotting terminating proper fractions. The cross-validated RMSE of the standard model (0.29 for plotting whole numbers, 0.46 for plotting pi) is close to what Bulut et al. (2023) report from pyBKT.

20.3 Example 2: Knowledge tracing and IRT

How does BKT compare with an IRT model on the same logs? `mv_kt_irt()` holds out the last 30% of each student's attempts on each skill, fits both models to the rest, and forecasts the held-out answers. The IRT model is a Rasch model with crossed random effects for students and items, fitted with lme4.

```
cmp <- mv_kt_irt(logs, item = "problem", skills = plot_skills, seed = 1)
cmp
#> <mv_kt_irt> (EXPERIMENTAL) Knowledge tracing vs IRT on the same logs
#> Forecasts of the last 30% of each student's attempts (6 skill(s)):
#>   BKT:       RMSE 0.451, accuracy 0.584, AUC 0.759
#>   IRT (1PL): RMSE 0.445, accuracy 0.633, AUC 0.723
#> (means over skills)
#> IRT item difficulty vs log BKT guess: Median correlation -0.89 over 6 skill(s)
tidy(cmp)
#> # A tibble: 12 x 6
#>    skill                              model n_test  rmse accuracy    auc
#>    <chr>                              <chr>  <int> <dbl>    <dbl>  <dbl>
```

```
#>  1 Plot whole number                    BKT       66 0.156    1     NA
#>  2 Plot whole number                    IRT ~     66 0.147    1     NA
#>  3 Plot terminating proper fraction     BKT      253 0.541    0.356  0.750
#>  4 Plot terminating proper fraction     IRT ~    253 0.497    0.601  0.752
#>  5 Plot imperfect radical               BKT      206 0.551    0.578  0.725
#>  6 Plot imperfect radical               IRT ~    206 0.517    0.578  0.745
#>  7 Plot non-terminating improper fract~ BKT      150 0.550    0.38   0.753
#>  8 Plot non-terminating improper fract~ IRT ~    150 0.515    0.48   0.721
#>  9 Plot pi                              BKT       21 0.371    0.905  0.842
#> 10 Plot pi                              IRT ~     21 0.420    0.762  0.816
#> 11 Plot decimal - thousandths           BKT       74 0.540    0.284  0.724
#> 12 Plot decimal - thousandths           IRT ~     74 0.573    0.378  0.583
```

The two models forecast about equally well. BKT ranks the answers a little better on average (a mean AUC of 0.76 against 0.72), because it updates after every answer, while IRT keeps each student's ability fixed. IRT has the lower RMSE and the higher accuracy for three of the skills, because its item difficulties capture differences between problems that the standard BKT model ignores. Neither model is right: Students learn and items differ, and each model captures one of the two.

The second half of the output connects the parameters.

```
tidy(cmp, "items")
#> # A tibble: 6 x 3
#>   skill                                n_items correlation
#>   <chr>                                  <int>       <dbl>
#> 1 Plot whole number                          9      -0.897
#> 2 Plot terminating proper fraction          31      -0.867
#> 3 Plot imperfect radical                    21      -0.888
#> 4 Plot non-terminating improper fraction    13      -0.602
#> 5 Plot pi                                   10      -0.910
#> 6 Plot decimal - thousandths                14      -0.944
```

For every skill, the IRT difficulty of a problem is strongly correlated with the log of its BKT guess, with a median of about -0.89: The easier the problem, the more often students who do not know the skill still get it right. Bulut et al. (2023) report correlations between 0.55 and 0.94 (in absolute value) for the same skills. When item guess and slip are estimated, BKT's guess parameter is an item difficulty in disguise.

20.4 Example 3: Practice on a knowledge graph

Chapter 19 described the Grade 5 fractions test as a knowledge graph. `fractions_logs` holds simulated practice logs on the same items: 400 students practise the six concepts in blocks, mostly in the curriculum order, until they answer three in a row correctly.

```
practice <- fractions_logs
kg <- fractions_kg
```

The logs have no skill column. With `kg`, `mv_kt_fit()` maps each item to the concepts it measures in the graph.

```
head(practice)
#> # A tibble: 6 x 4
#>   student attempt item  correct
#>   <chr>     <int> <chr>   <int>
#> 1 P001          1 F01         0
#> 2 P001          2 F02         1
#> 3 P001          3 F04         1
#> 4 P001          4 F02         1
#> 5 P001          5 F11         0
#> 6 P001          6 F10         1
kt_fr <- mv_kt_fit(practice, kg = kg, seed = 1)
kt_fr
#> <mv_kt> (EXPERIMENTAL) Bayesian knowledge tracing
#> 6 skill(s), 400 student(s), 20338 attempt(s)
#>                skill prior learn guess  slip
#>              meaning 0.473 0.314 0.213 0.105
#>           equivalent 0.128 0.136 0.173 0.116
#>                mixed 0.072 0.174 0.176 0.101
#>              compare 0.027 0.116 0.193 0.033
#>  common_denominator 0.015 0.118 0.241 0.080
#>         add_subtract 0.001 0.072 0.198 0.114
#> Fit on the same logs: RMSE 0.452, accuracy 0.695, AUC 0.718
```

```
gg_kt(kt_fr, type = "graph")
```

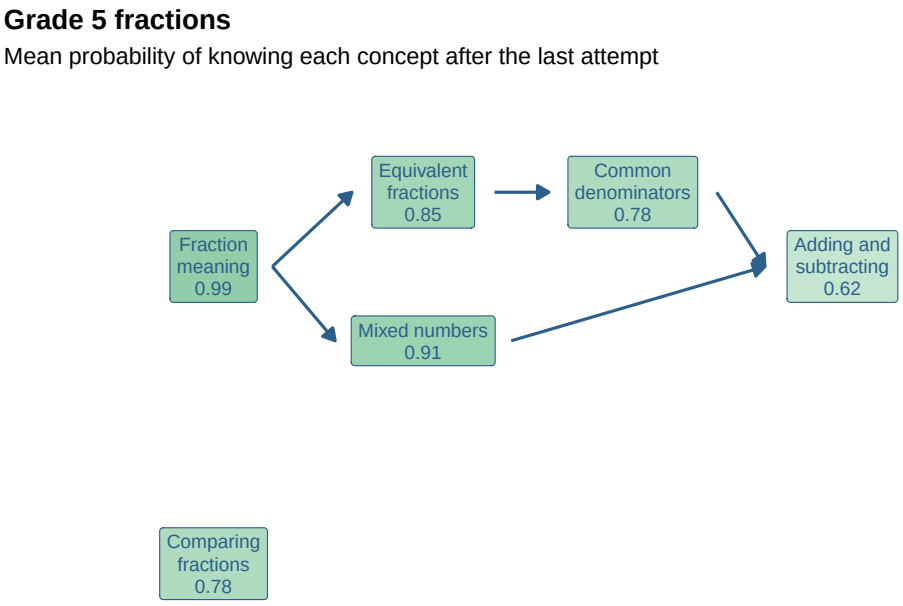


Figure 20.3: The fractions graph shaded by the mean probability of knowing each concept after the last practice attempt.

The graph view shows where the class stands: Nearly everyone has fraction meaning, and adding and subtracting is the least secure. For one student, `gg_kt(kt_fr, student = "P001", type = "graph")` gives the same picture, a diagnostic report that follows the graph.

20.4.1 Checking prerequisites against the logs

A prerequisite is a claim about learning: A student who does not know equivalent fractions should struggle to learn to add fractions with unlike denominators. Practice logs can test that claim directly. `mv_kt_prereq()` estimates, for every pair of concepts A and B, the learn rate of B on the attempts made while A was probably known and on those made while it was not, and tests the difference with a likelihood ratio test. Each test uses only the attempts where B's other prerequisites in the graph are already known, so an indirect prerequisite adds little.

```
check <- mv_kt_prereq(kt_fr)
check
#> <mv_kt_prereq> (EXPERIMENTAL) Prerequisites checked against practice logs
#> Graph: Grade 5 fractions; 5 prerequisite edge(s), 5 supported by the logs
#> 1 flag(s)
#> * [missing_prerequisite] compare is learned 10.1 times as fast once
#>   equivalent is known (learn 0.17 vs 0.02, LR = 155.6); add the
#>   prerequisite equivalent -> compare?
head(tidy(check)[, c("from", "to", "edge", "learn_without", "learn_with", "ratio",
 ↪  "lr", "verdict")], 8)
```

```
#> # A tibble: 8 x 8
#>   from           to    edge  learn_without learn_with ratio     lr verdict
#>   <chr>          <chr> <lgl>         <dbl>      <dbl> <dbl>  <dbl> <chr>
#> 1 equivalent     comp~ FALSE        0.0173      0.175 10.1  156.   "missi~
#> 2 common_denomi~ add_~ TRUE         0.0263      0.127  4.82  96.6  "suppo~
#> 3 meaning        equi~ TRUE         0.0620      0.211  3.39  89.4  "suppo~
#> 4 equivalent     comm~ TRUE         0.0186      0.153  8.22  88.4  "suppo~
#> 5 meaning        mixed TRUE         0.0594      0.202  3.40  44.0  "suppo~
#> 6 mixed          add_~ TRUE         0.0196      0.127  6.46  23.9  "suppo~
#> 7 common_denomi~ mixed FALSE        0.195       0.430  2.20  11.1  ""
#> 8 equivalent     add_~ FALSE        0.0393      0.133  3.40   9.69 ""
```

All five prerequisites of the graph are supported: Each concept is learned three to eight times as fast once its prerequisite is known. The strongest effect in the logs, though, is not in the graph: Comparing fractions is learned ten times as fast once equivalent fractions are known. This is the edge the published graph misses, the same one that `mv_kg_fit()` found in the test answers in Chapter 19. Two independent kinds of evidence, a test taken once and practice over time, point to the same repair.

```
kg2 <- mv_kg_edit(kg, add = data.frame(from = "equivalent", to = "compare"))
mv_kt_prereq(kt_fr, kg = kg2)
#> <mv_kt_prereq> (EXPERIMENTAL) Prerequisites checked against practice logs
#> Graph: Grade 5 fractions; 6 prerequisite edge(s), 6 supported by the logs
#> No flags: The logs agree with the graph's prerequisites.
```

With the edge added, the logs agree with the graph. Practice logs have one more advantage over a test: The direction of a prerequisite is in the data. Equivalent fractions speed up learning to compare, and not the other way round, so no assumption about which concept is easier is needed.

The learn rates of the first model also make sense now. They are lower than the rates the logs were simulated with (for example 0.12 against 0.20 for comparing fractions), because some students practised a concept before its prerequisites and learned it slowly. A single learn rate per skill averages over both groups; the graph explains why learning is uneven.

20.5 Your turn

1. Fit the standard model to all 12 skills of `cognitive_tutor`. Which skill has the lowest learn rate, and what would you tell the tutor's designers about it?
2. Refit the plotting skills with both `forgets = TRUE` and `multigs = TRUE`. Does forgetting help once items have their own guess and slip?
3. Run `mv_kt_irt()` on the three "Finding the intersection" skills. Which model forecasts better, and why might that be?

4. Add a wrong prerequisite, `mixed -> compare`, to the fractions graph (together with `equivalent -> compare`) and run `mv_kt_prereq()`. What does it report?

Solutions

```
# 1. all skills
kt_all <- mv_kt_fit(logs, seed = 1)
p <- tidy(kt_all)
p[order(p$learn), c("skill", "prior", "learn", "guess", "slip")][1:3, ]
#> # A tibble: 3 x 5
#>   skill                          prior  learn guess   slip
#>   <chr>                          <dbl>  <dbl> <dbl>  <dbl>
#> 1 Plot terminating proper fraction 0.574 0.0266 0.318 0.328
#> 2 Finding the intersection, Mixed  0.430 0.0503 0.338 0.324
#> 3 Finding the intersection, GLF    0.128 0.0543 0.466 0.0103

# 2. both variants
kt_fg <- mv_kt_fit(logs, item = "problem", skills = plot_skills, multigs = TRUE,
                   forgets = TRUE, seed = 1)
rbind(multigs = glance(kt_g), both = glance(kt_fg))
#> # A tibble: 2 x 9
#>   n_skills n_students n_attempts loglik   AIC   BIC  rmse accuracy   auc
#> *    <int>      <int>      <int>  <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>
#> 1        6        258       4336 -2313. 5054. 6418. 0.424    0.726 0.805
#> 2        6        258       4336 -2313. 5066. 6469. 0.424    0.726 0.805

# 3. intersection skills
inter <- grep("^Finding", unique(logs$skill), value = TRUE)
mv_kt_irt(logs, item = "problem", skills = inter, seed = 1)
#> <mv_kt_irt> (EXPERIMENTAL) Knowledge tracing vs IRT on the same logs
#> Forecasts of the last 30% of each student's attempts (3 skill(s)):
#>   BKT:       RMSE 0.493, accuracy 0.573, AUC 0.585
#>   IRT (1PL): RMSE 0.440, accuracy 0.715, AUC 0.765
#> (means over skills)
#> IRT item difficulty vs log BKT guess: Median correlation -0.89 over 3 skill(s)

# 4. a wrong prerequisite
kg3 <- mv_kg_edit(kg, add = data.frame(from = c("equivalent", "mixed"),
                                       to = c("compare", "compare")))
mv_kt_prereq(kt_fr, kg = kg3)
#> <mv_kt_prereq> (EXPERIMENTAL) Prerequisites checked against practice logs
#> Graph: Grade 5 fractions; 7 prerequisite edge(s), 6 supported by the logs
#> 1 flag(s)
#> * [weak_prerequisite] The logs show little effect of mixed on learning
```

```
#>   compare (learn 0.17 vs 0.17, p = 0.90); is mixed -> compare a
#>   prerequisite?
```

1. Plotting terminating proper fractions has the lowest learn rate (about 0.03): Students barely improve with practice. The designers should look at the feedback and hints for this skill, or split it into smaller skills; it may also be two skills tagged as one, which BKT would see as slow learning with many slips.
2. No: The log-likelihood is the same and AIC rises. Wrong answers by students who seemed to know a skill are explained by item slips, not by forgetting.
3. IRT forecasts better for these skills. Each has few students (about 70) and items that differ a lot in difficulty, which the Rasch model captures and the standard BKT model does not; with sparse sequences there is also little learning for BKT to track.
4. It flags `mixed -> compare` as a weak prerequisite: Comparing fractions is learned just as fast whether or not a student knows mixed numbers. The check works in both directions, finding missing edges and questioning ones that are there.

Further reading

Corbett and Anderson (1995) introduced knowledge tracing in the ACT Programming Tutor. Badrinath, Wang and Pardos (2021) describe pyBKT and its variants, and Bulut et al. (2023) introduce BKT for readers who know IRT, with the examples this chapter follows. Pardos and Heffernan (2011) add item difficulty to knowledge tracing, and Deonovic et al. (2018) work out the formal relation between BKT and IRT. Deep knowledge tracing (Piech et al. 2015) replaces the hidden Markov model with a recurrent neural network; it needs much larger logs, and its predictions are harder to interpret.

Appendix A

Datasets used in the book

Every example loads its data in one chunk, labelled `data-<chapter>-<n>`, at the start of the example. The table lists what each chunk defines. *Real* data come with mirt; *simulated* data are generated in the chunk itself, so the true parameters are known and the examples can check estimates against them.

Table A.1: Data chunks in the book and the objects each defines.

Chunk	Data	Kind	Defines
`data-measurement-1`	`SAT12`: 600 students, 32 five-option science items	Real	`raw`, `key`
`data-measurement-2`	8-item wellbeing scale, 0 to 3, two negatively worded items	Simulated	`resp`
`data-irt-models-1`	`LSAT7`: 1,000 test takers, 5 items	Real	`resp`
`data-irt-models-2`	20-item, four-option driving theory test with guessing	Simulated	`resp`, `true`
`data-estimation-1`	`LSAT6`: 1,000 test takers, 5 items	Real	`resp`
`data-estimation-2`	`SAT12`, scored with the corrected key	Real	`resp`, `key`
`data-dimensionality-1`	24-item mathematics test with calculation and reading dimensions	Simulated	`resp`, `a`, `d`, `sigma`

Chunk	Data	Kind	Defines
`data-dimensionality-2`	20-item reading test with a four-item passage	Simulated	`resp`
`data-fit-1`	`SAT12`, scored with the corrected key	Real	`resp`, `key`
`data-fit-2`	30-item quiz with random responders and pre-knowledge	Simulated	`resp`, `type`, `a`, `b`
`data-information-1`	`SAT12`, scored with the corrected key	Real	`resp`, `key`
`data-information-2`	Item parameters of a 90-item nursing pool	Simulated	`pool`
`data-scoring-1`	`LSAT7`	Real	`resp`
`data-scoring-2`	40-item mathematics test, true abilities known	Simulated	`resp`, `true_theta`
`data-polytomous-1`	`Science`: 392 respondents, 4 attitude items scored 1 to 4	Real	`resp`
`data-polytomous-2`	6-item job satisfaction scale, 0 to 4, one item with disordered steps	Simulated	`resp`
`data-dif-1`	25-item reading test taken on paper or tablet	Simulated	`resp`, `group`
`data-dif-2`	20-item chemistry quiz by gender	Simulated	`resp`, `group`
`data-linking-1`	Two 30-item placement forms with 10 anchors (NEAT design)	Simulated	`old`, `new`, `anchors`, `pars`
`data-linking-2`	The same two forms stacked for concurrent calibration	Simulated	`resp`, `form`
`data-banks-cat-1`	Parameters of a 200-item vocabulary bank in four areas	Simulated	`bank`
`data-banks-cat-2`	300 simulated test takers for the CAT	Simulated	`true_theta`

Chunk	Data	Kind	Defines
`data-reporting-1`	36-item Grade 6 science test	Simulated	`resp`
`data-reporting-2`	`Science`	Real	`resp`
`data-ml-lens-1`	20-item test with known abilities and item parameters	Simulated	`resp`, `theta`, `a`, `b`
`data-ml-lens-2`	40-item placement test and a pass/fail course result	Simulated	`resp`, `ability`, `passed`
`data-item-text-1`	70 science classification items with text features and the responses of 1,000 students	Simulated	`sci`, `resp`, `rasch`
`data-aig-1`	Three item models for Grade 7 items: Average speed, a fraction of a number and the mean	Simulated	`speed`, `frac`, `avg`, `fams`
`data-aig-2`	A field test of the 36 generated items, answered by 800 students	Simulated	`resp`, `rasch`, `items`
`data-ai-validity-1`	Machine and human scores of two writing prompts, by device and testing window	Simulated	`writing`
`data-ai-validity-2`	Field-test difficulties and features of 45 items from three item models	Simulated	`items`
`data-auto-scoring-1`	Short answers to a science question, scored 0 to 3 by one or two raters	Simulated	`bread`
`data-auto-scoring-2`	Four answers with a known correct score	Simulated	`probe`

Chunk	Data	Kind	Defines
`data-auto-scoring-3`	A test with 10 multiple-choice items and three written answers scored by two raters and an engine	Simulated	`written`, `mc`
`data-llms-1`	`Science`	Real	`resp`
`data-llms-2`	A two-cell blueprint for a numeracy quiz	Simulated	`cells`
`data-llms-3`	`hexaco_real` and `hexaco_llm`: 40 Honesty-Humility items, real Canadian respondents and four language models (4 attempts each)	Real and generated	`real`, `synthetic`
`data-knowledge-graphs-1`	`fraction_subtraction`: 536 students, 20 fraction subtraction items, with its 8-attribute Q-matrix as a knowledge graph	Real	`resp`, `kg`
`data-knowledge-tracing-1`	`cognitive_tutor`: Cognitive Tutor practice logs, 587 students, 16,857 attempts on 12 skills	Real	`logs`
`data-knowledge-tracing-2`	`fractions_logs` and `fractions_kg`: Practice logs of 400 students on the fractions graph	Simulated	`practice`, `kg`

The book deliberately does not analyse the mirtverse lab datasets (`certexam`, `numeracy`, `linking`, `catbank` and `openended`), because the labs ask students to discover problems hidden in them. The same goes for the AI Hub datasets `itemtext` and `essays`, and the knowledge graph example `fractions`. They are described in the teaching labs article and on their help pages, for example `?certexam`.

Other datasets for practice

mirt ships more datasets that work well for exercises:

- `LSAT6` and `LSAT7`: Short dichotomous tests (expand them with `expand.table()`).
- `deAyala`: Five dichotomous mathematics items from de Ayala's textbook, as a frequency table.
- `SLF`: Five dichotomous statements about social life feelings, answered by 1,490 respondents.
- `Bock1997`: A three-item polytomous test, as a frequency table.
- `ASVAB`: Four arithmetic reasoning items with response counts for four groups defined by sex and race, a ready-made exercise for multiple-group models and DIF.

Use `?<name>` for the details of each.

Appendix B

Adapting the book for your course

The book is meant to be changed. Each chapter is a separate R Markdown file, each example starts from one data chunk, and the book builds with one command, so you can reorder chapters, replace examples with your own data, or add a chapter for a topic you teach.

The files

All sources are in the `book/` folder of the mirtverse repository:

Table B.1: Source files of the book.

File	Contents
`index.Rmd`	Title page and preface
`00-author.Rmd`	About the author
`01-measurement.Rmd` to `12-reporting.Rmd`	The twelve chapters, one per course week
`13-ml-lens.Rmd`, `14-item-text.Rmd`, `15-aig.Rmd`, `16-auto-scoring.Rmd`, `17-llms.Rmd`, `18-ai-validity.Rmd`	The chapters on AI in measurement
`90-datasets.Rmd`, `91-adapting.Rmd`	The appendices
`_bookdown.yml`	Chapter order (`rmd_files`) and output folder
`_output.yml`	Look of the HTML, PDF and EPUB editions
`latex/`	Title page, preamble and box styles of the PDF edition
`cover/`	Cover artwork and how to regenerate it
`_common.R`	Code run before every chapter: Loads mirtverse and sets chunk options
`references.bib`	References cited in the book

The chapter files and their ids (`{#measurement}`, `{#irt-models}` and so on) are kept stable, so links to a chapter keep working across versions.

Every chapter has the same shape

1. A **Learning goals** box.
2. **Key terms**, linked to the glossary.
3. **The ideas.**
4. **Example 1** and **Example 2**, each starting with a data chunk labelled `data-<chapter id>-1` or `-2`.
5. **Your turn**, with solutions in a folded `<details>` block.
6. **In the lab** and **Further reading**.

Keep this shape when you add a chapter, and students will always know where to look.

Swapping in your own data

To use your own data in an example, replace the body of its data chunk and keep the label and the object names listed in Appendix A. For example, to run Example 1 of Chapter 2 on your own dichotomous test:

````
```{r data-irt-models-1}
# Our midterm: 40 multiple-choice items, scored 0/1.
resp <- read.csv("midterm-scored.csv")[, -1]   # drop the ID column
```
````

The rest of the example then runs on your data. Two caveats:

- Examples built on **simulated** data compare estimates with the true values used to simulate them (objects such as `true`, `true_theta` or the planted DIF items). With real data there is no truth, so delete those comparisons or turn them into questions for students.
- The prose describes the original results. Re-read the text around each output after you swap the data, and edit the interpretation to match.

The mirtverse instructor toolkit automates these steps, for example copying a chapter into a course folder and replacing a data chunk; see the instructor toolkit guide.

Building the book

Install bookdown once, then build from the `book/` folder:

```
install.packages("bookdown")
bookdown::render_book("index.Rmd")           # the whole book, into _book/
bookdown::preview_chapter("05-fit.Rmd")      # one chapter, faster
```

Open `_book/index.html` in a browser to read the result. The whole book builds in a few minutes on a laptop. To publish your own version, put the `_book/` folder on any web server, or use GitHub Pages as the mirtverse repository does (its `book` workflow builds the book and publishes it next to the package website).

Suggesting changes

Every page has a *Suggest an edit* link at the top, which opens the chapter's source on GitHub. Corrections and new examples are welcome as issues or pull requests in the mirtverse repository.

AlKhuzaey, Samah, Floriana Grasso, Terry R. Payne, and Valentina Tamma. 2024. "Text-Based Question Difficulty Prediction: A Systematic Review of Automatic Approaches." *International Journal of Artificial Intelligence in Education* 34 (3): 862–914. https://doi.org/10.1007/s40593-023-00362-1.

American Educational Research Association, American Psychological Association, and National Council on Measurement in Education. 2014. *Standards for Educational and Psychological Testing*. American Educational Research Association.

Argyle, Lisa P., Ethan C. Busby, Nancy Fulda, Joshua R. Gubler, Christopher Rytting, and David Wingate. 2023. "Out of One, Many: Using Language Models to Simulate Human Samples." *Political Analysis* 31 (3): 337–51. https://doi.org/10.1017/pan.2023.2.

Ashton, Michael C., Kibeom Lee, and Lewis R. Goldberg. 2007. "The IPIP–HEXACO Scales: An Alternative, Public-Domain Measure of the Personality Constructs in the HEXACO Model." *Personality and Individual Differences* 42 (8): 1515–26. https://doi.org/10.1016/j.paid.2006.10.027.

Ayala, R. J. de. 2022. *The Theory and Practice of Item Response Theory*. 2nd ed. Guilford Press.

Badrinath, Anirudhan, Frederic Wang, and Zachary A. Pardos. 2021. "pyBKT: An Accessible Python Library of Bayesian Knowledge Tracing Models." *Proceedings of the 14th International Conference on Educational Data Mining*, 468–74.

Benedetto, Luca, Paolo Cremonesi, Andrew Caines, et al. 2023. "A Survey on Recent Approaches to Question Difficulty Estimation from Text." *ACM Computing Surveys* 55 (9): Article 178. https://doi.org/10.1145/3556538.

Bock, R. Darrell, and Murray Aitkin. 1981. "Marginal Maximum Likelihood Estimation of Item Parameters: Application of an EM Algorithm." *Psychometrika* 46 (4): 443–59.

Breiman, Leo. 2001. "Statistical Modeling: The Two Cultures." *Statistical Science* 16 (3): 199–231. https://doi.org/10.1214/ss/1009213726.

Bridgeman, Brent, Catherine Trapani, and Yigal Attali. 2012. "Comparison of Human and Machine Scoring of Essays: Differences by Gender, Ethnicity, and Country." *Applied Measurement in Education* 25 (1): 27–40. https://doi.org/10.1080/08957347.2012.635502.

Bulut, Okan, Maggie Beiting-Parrish, Jodi M. Casabianca, et al. 2024. "The Rise of Artificial Intelligence in Educational Measurement: Opportunities and Ethical Challenges." *Chinese/English Journal of Educational Measurement and Evaluation*, ahead of print. https://doi.org/10.59863/MIQL7785.

Bulut, Okan, Guher Gorgun, and Seyma Nur Yildirim-Erbasli. 2021. "Estimating Explanatory Extensions of Dichotomous and Polytomous Rasch Models: The Eirm Package in R." *Psych* 3 (3): 308–21. https://doi.org/10.3390/psych3030023.

Bulut, Okan, Jinnie Shin, Seyma N. Yildirim-Erbasli, Guher Gorgun, and Zachary A. Pardos. 2023. "An Introduction to Bayesian Knowledge Tracing with pyBKT." *Psych* 5 (3): 770–86. https://doi.org/10.3390/psych5030050.

Chalmers, R. Philip. 2012. "mirt: A Multidimensional Item Response Theory Package for the R Environment." *Journal of Statistical Software* 48 (6): 1–29. https://doi.org/10.18637/jss.v048.i06.

Cizek, Gregory J., and Michael B. Bunch. 2007. *Standard Setting: A Guide to Establishing and Evaluating Performance Standards on Tests*. Sage.

Cohen, Jacob. 1968. "Weighted Kappa: Nominal Scale Agreement with Provision for Scaled Disagreement or Partial Credit." *Psychological Bulletin* 70 (4): 213–20.

Corbett, Albert T., and John R. Anderson. 1995. "Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge." *User Modeling and User-Adapted Interaction* 4 (4): 253–78. https://doi.org/10.1007/BF01099821.

Crocker, Linda, and James Algina. 1986. *Introduction to Classical and Modern Test Theory*. Holt, Rinehart; Winston.

De Boeck, Paul, and Mark Wilson, eds. 2004. *Explanatory Item Response Models: A Generalized Linear and Nonlinear Approach*. Springer. https://doi.org/10.1007/978-1-4757-3990-9.

DeCarlo, Lawrence T. 2011. "On the Analysis of Fraction Subtraction Data: The DINA Model, Classification, Latent Class Sizes, and the Q-Matrix." *Applied Psychological Measurement* 35 (1): 8–26.

Deonovic, Benjamin, Michael Yudelson, Maria Bolsinova, Meirav Attali, and Gunter Maris. 2018. "Learning Meets Assessment: On the Relation Between Item Response Theory and Bayesian Knowledge Tracing." *Behaviormetrika* 45 (2): 457–74. https://doi.org/10.1007/s41237-018-0070-z.

Doignon, Jean-Paul, and Jean-Claude Falmagne. 1999. *Knowledge Spaces*. Springer.

Drasgow, Fritz, Michael V. Levine, and Esther A. Williams. 1985. "Appropriateness Measurement with Polychotomous Item Response Models and Standardized Indices." *British Journal of Mathematical and Statistical Psychology* 38 (1): 67–86.

Embretson, Susan E., and Steven P. Reise. 2000. *Item Response Theory for Psychologists*. Lawrence Erlbaum Associates.

Fischer, Gerhard H. 1973. "The Linear Logistic Test Model as an Instrument in Educational Research." *Acta Psychologica* 37 (6): 359–74. https://doi.org/10.1016/0001-6918(73)90003-6.

Geerlings, Hanneke, Cees A. W. Glas, and Wim J. van der Linden. 2011. "Modeling Rule-Based Item Generation." *Psychometrika* 76 (2): 337–59. https://doi.org/10.1007/s11336-011-9204-x.

Gierl, Mark J., and Thomas M. Haladyna, eds. 2013. *Automatic Item Generation: Theory and Practice*. Routledge.

Gierl, Mark J., and Hollis Lai. 2013. "Instructional Topics in Educational Measurement (ITEMS) Module: Using Automated Processes to Generate Test Items." *Educational Measurement: Issues and Practice* 32 (3): 36–50. https://doi.org/10.1111/emip.12018.

Glas, Cees A. W., and Wim J. van der Linden. 2003. "Computerized Adaptive Testing with Item Cloning." *Applied Psychological Measurement* 27 (4): 247–61. https://doi.org/10.1177/0146621603027004001.

Hambleton, Ronald K., H. Swaminathan, and H. Jane Rogers. 1991. *Fundamentals of Item Response Theory*. Sage.

Hao, Jiangang, Alina A. von Davier, Victoria Yaneva, Susan Lottridge, Matthias von Davier, and Deborah J. Harris. 2024. "Transforming Assessment: The Impacts and Implications of Large Language Models and Generative AI." *Educational Measurement: Issues and Practice* 43 (2): 16–29. https://doi.org/10.1111/emip.12602.

Holland, Paul W., and Dorothy T. Thayer. 1988. "Differential Item Performance and the Mantel-Haenszel Procedure." In *Test Validity*, edited by Howard Wainer and Henry I. Braun. Lawrence Erlbaum Associates.

Holland, Paul W., and Howard Wainer, eds. 1993. *Differential Item Functioning*. Lawrence Erlbaum Associates.

International Test Commission, and Association of Test Publishers. 2022. *Guidelines for Technology-Based Assessment*.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2021. *An Introduction to Statistical Learning with Applications in R*. 2nd ed. Springer. https://doi.org/10.1007/978-1-0716-1418-1.

Johnson, Matthew S., Xiang Liu, and Daniel F. McCaffrey. 2022. "Psychometric Methods to Evaluate Measurement and Algorithmic Bias in Automated Scoring." *Journal of Educational Measurement* 59 (3): 338–61. https://doi.org/10.1111/jedm.12335.

Kane, Michael T. 2013. "Validating the Interpretations and Uses of Test Scores." *Journal of Educational Measurement* 50 (1): 1–73. https://doi.org/10.1111/jedm.12000.

Kolen, Michael J., and Robert L. Brennan. 2014. *Test Equating, Scaling, and Linking: Methods and Practices*. 3rd ed. Springer.

Leighton, Jacqueline P., Mark J. Gierl, and Stephen M. Hunka. 2004. "The Attribute Hierarchy Method for Cognitive Assessment: A Variation on Tatsuoka's Rule-Space Approach." *Journal of Educational Measurement* 41 (3): 205–37.

Linacre, John M. 1989. *Many-Facet Rasch Measurement*. MESA Press.

Linden, Wim J. van der. 2005. *Linear Models for Optimal Test Design*. Springer.

Lord, Frederic M. 1980. *Applications of Item Response Theory to Practical Testing Problems*. Lawrence Erlbaum Associates.

Loukina, Anastassia, Nitin Madnani, Aoife Cahill, et al. 2020. "Using PRMSE to Evaluate Automated Scoring Systems in the Presence of Label Noise." *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 18–29. https://aclanthology.org/2020.bea-1.2.

Masters, Geoff N. 1982. "A Rasch Model for Partial Credit Scoring." *Psychometrika* 47 (2): 149–74.

Maydeu-Olivares, Alberto, and Harry Joe. 2006. "Limited Information Goodness-of-Fit Testing in Multidimensional Contingency Tables." *Psychometrika* 71 (4): 713–32.

Muraki, Eiji. 1992. "A Generalized Partial Credit Model: Application of an EM Algorithm." *Applied Psychological Measurement* 16 (2): 159–76.

Orlando, Maria, and David Thissen. 2000. "Likelihood-Based Item-Fit Indices for Dichotomous Item Response Theory Models." *Applied Psychological Measurement* 24 (1): 50–64.

Pardos, Zachary A., and Neil T. Heffernan. 2011. "KT-IDEM: Introducing Item Difficulty to

the Knowledge Tracing Model." *User Modeling, Adaption and Personalization*, 243–54. https://doi.org/10.1007/978-3-642-22362-4_21.

Piech, Chris, Jonathan Bassen, Jonathan Huang, et al. 2015. "Deep Knowledge Tracing." *Advances in Neural Information Processing Systems* 28.

Rudner, Lawrence M. 2001. "Computing the Expected Proportions of Misclassified Examinees." *Practical Assessment, Research, and Evaluation* 7 (14).

Samejima, Fumiko. 1969. "Estimation of Latent Ability Using a Response Pattern of Graded Scores." *Psychometrika Monograph Supplement* 34 (4, Pt. 2).

Sinharay, Sandip, Matthew S. Johnson, and David M. Williamson. 2003. "Calibrating Item Families and Summarizing the Results Using Family Expected Response Functions." *Journal of Educational and Behavioral Statistics* 28 (4): 295–313.

Stamper, John, Alexandru Niculescu-Mizil, Steven Ritter, Geoffrey J. Gordon, and Kenneth R. Koedinger. 2010. *Challenge Data Sets from KDD Cup 2010*. PSLC DataShop. https://pslcdatashop.web.cmu.edu/KDDCup/.

Stocking, Martha L., and Frederic M. Lord. 1983. "Developing a Common Metric in Item Response Theory." *Applied Psychological Measurement* 7 (2): 201–10.

Tatsuoka, Curtis. 2002. "Data Analytic Methods for Latent Partially Ordered Classification Models." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 51 (3): 337–50.

Tatsuoka, Kikumi K. 1984. *Analysis of Errors in Fraction Addition and Subtraction Problems*. Final Report NIE-G-81-0002. University of Illinois, Computer-based Education Research Laboratory.

Templin, Jonathan, and Laine Bradshaw. 2014. "Hierarchical Diagnostic Classification Models: A Family of Models for Estimating and Testing Attribute Hierarchies." *Psychometrika* 79 (2): 317–39.

Torre, Jimmy de la, and Jeffrey A. Douglas. 2004. "Higher-Order Latent Trait Models for Cognitive Diagnosis." *Psychometrika* 69 (3): 333–53.

Wainer, Howard et al. 2000. *Computerized Adaptive Testing: A Primer*. 2nd ed. Lawrence Erlbaum Associates.

Warm, Thomas A. 1989. "Weighted Likelihood Estimation of Ability in Item Response Theory." *Psychometrika* 54 (3): 427–50.

Williamson, David M., Xiaoming Xi, and F. Jay Breyer. 2012. "A Framework for Evaluation and Use of Automated Scoring." *Educational Measurement: Issues and Practice* 31 (1): 2–13. https://doi.org/10.1111/j.1745-3992.2011.00223.x.

Yan, Duanli, André A. Rupp, and Peter W. Foltz, eds. 2020. *Handbook of Automated Scoring: Theory into Practice*. CRC Press.

Yaneva, Victoria, Kai North, Peter Baldwin, et al. 2024. "Findings from the First Shared Task on Automated Prediction of Difficulty and Response Time for Multiple-Choice Questions." *Proceedings of the 19th Workshop on Innovative Use of NLP for Building Educational Applications*, 470–82. https://aclanthology.org/2024.bea-1.39.

Yarkoni, Tal, and Jacob Westfall. 2017. "Choosing Prediction over Explanation in Psychology: Lessons from Machine Learning." *Perspectives on Psychological Science* 12 (6): 1100–1122. https://doi.org/10.1177/1745691617693393.

Yen, Wendy M. 1984. "Effects of Local Item Dependence on the Fit and Equating Performance of the Three-Parameter Logistic Model." *Applied Psychological Measurement* 8 (2): 125–45.