Reject inference with nested conditional models based on joint risk and fraud scores

Ross Gayler

CSCC 28/8/2013

Prologue

- Combination of case study / natural history study / soap box

- Not about reject inference in isolation
- About reject inference in a realistic context
  - Real problem typically more complex than text-book examples
    - More outcomes
    - More censoring processes
    - Interacting commercial and pragmatic constraints

- Emphasis on craft of credit scoring
  - Many moving parts to be integrated
  - Value of conceptual simplicity
  - Designing components for re-use and integration
  - Not time to consider all design issues

veda
applied intelligence

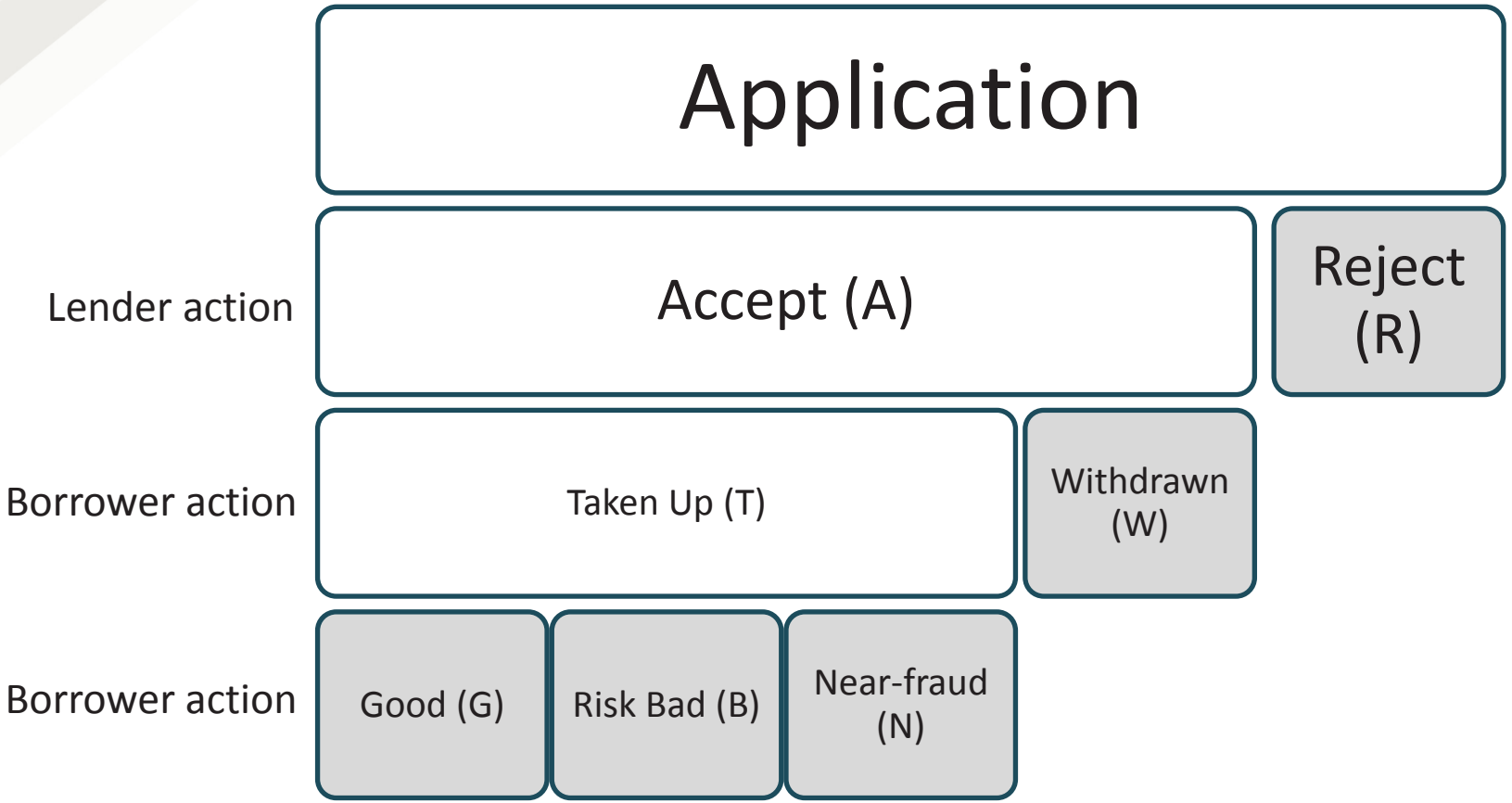
Case study problem setting

Determine the decision boundaries

- Application scoring
- Two negative outcomes: Risk Bad (B), Near-fraud (N)
  - Mutually exclusive by definition
  - Might or might not be “competing” processes
  - Fraud may have been the intent at time of application
- Two application scorecards were developed (risk and fraud)
- Scores to be used jointly in decision-making (matrix approach)
- Lender needs to set decision boundaries in joint score matrix
  - Need to populate joint score matrix with historical outcomes
  - Need historical outcomes to be as though all are accepted
    - Determine which applications ought not have been accepted
  - Some historical outcomes are necessarily censored
    - B or N only seen if accepted by lender and taken up by applicant
  - Therefore, need to infer outcomes for rejected applications

Expanded range of outcomes

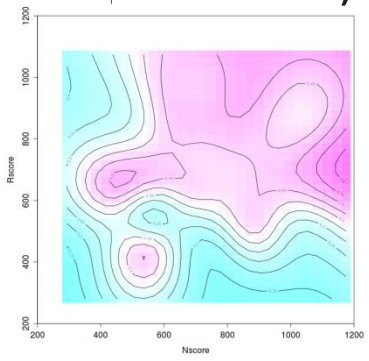
- There are more outcomes than the standard Reject, Good, and Bad
- Multiple nested outcomes (conditional on parent outcome)



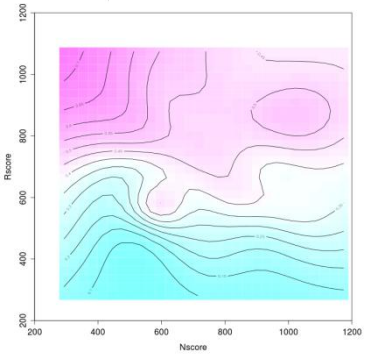
Joint risk and fraud scores

- Multiple independent behavioural dimensions
  - Need multiple predictions (scores)
- Outcome probabilities may be interactive functions of the two scores
  - Very likely because the outcomes are mutually exclusive and more outcomes than dimensions
  - Exploiting that interaction yields a more predictive system
- Optimal decision boundaries follow outcome contours in score space
  - Interaction in predictions implies interaction in decision boundary

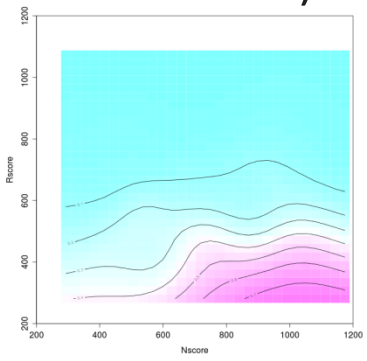
Withdrawn (W)



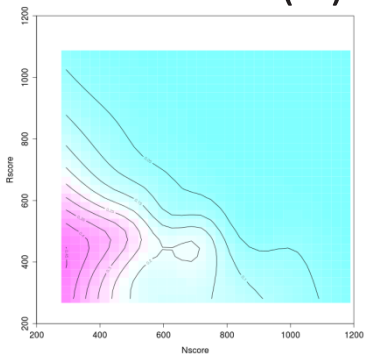
Good (G)



Risk Bad (B)



Near-fraud (N)



veda
applied intelligence

Design issues

Complexity of joint score approach

Complexity of joint score approach

Problems

- **Cost**: Joint scores can take more time to build & cost more to implement
- **Project risk**: Joint scores may not yield benefits or may not be buildable

Solutions

- Develop joint score components as independent add-ons
  - Two traditional full models built: Risk (Rscore) & Near-fraud (Nscore)
  - Joint use doesn't impact development of Rscore and Nscore
  - Can always drop back to using the scores separately
- Use inexpensive joint score modelling process based on calibration
- Use inexpensive joint score implementation process based on calibration

Design issues

Scores to be used jointly

Scores to be used jointly (1)

Problem

- Known that B and N are quite similar
  - G vs B and G vs N scores would be highly correlated
  - Joint score matrix would be populated on the diagonal only
- Need scores to be relatively independent (so joint score matrix well filled)
- While maintaining compatibility with separate use (fall-back position)

Solution

- Rscore = G vs B (standard model: classed predictors, logistic regression)
- Nscore = B vs N (standard model: classed predictors, logistic regression)

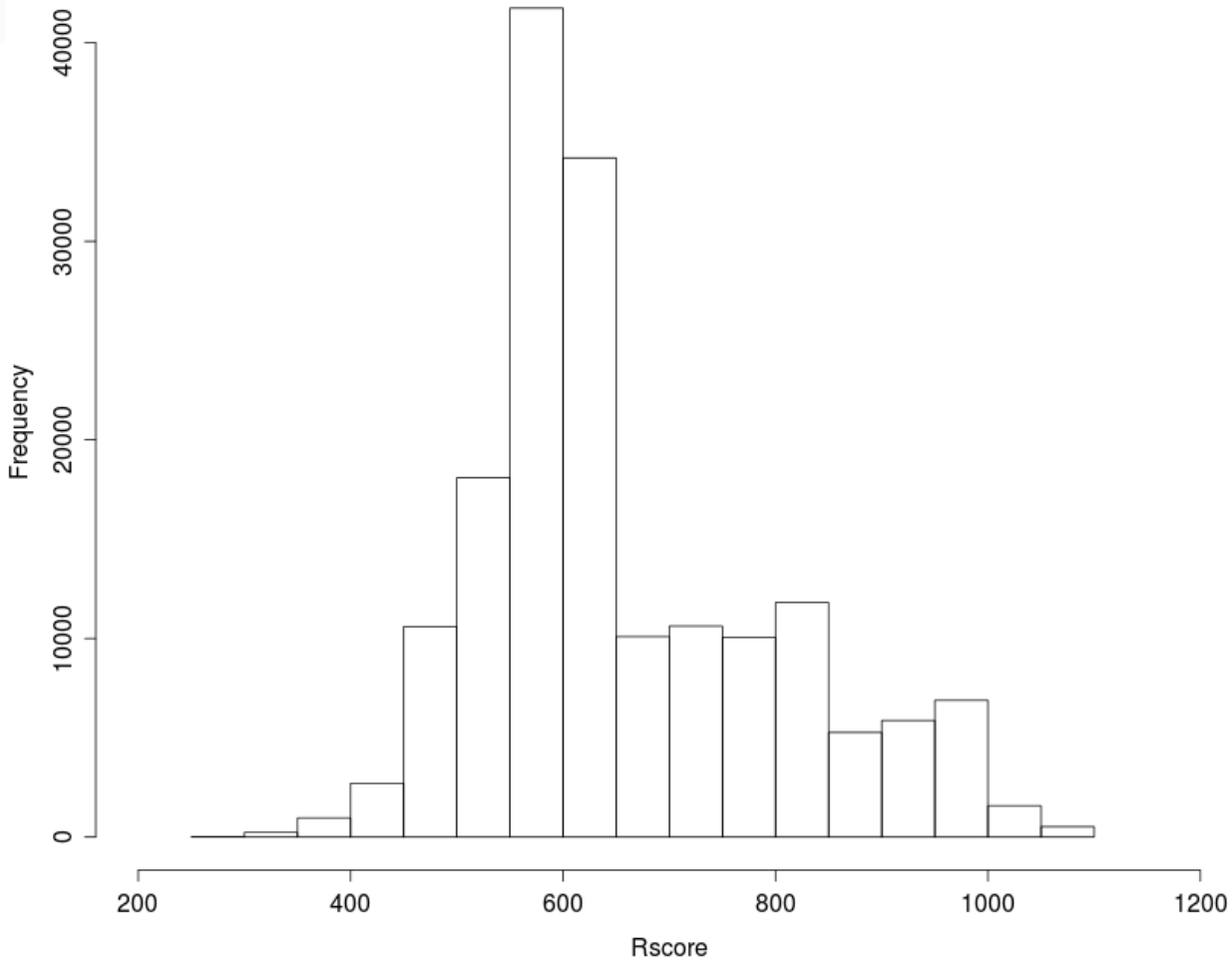
Scores to be used jointly (2)

Solution

- Rscore = G vs B
  - Conceptually simple
  - Useable as a fall-back model if Nscore didn't pan out
    - If B indistinguishable from N
      - Can't build Nscore
      - No need for Nscore
      - No harm done to Rscore by excluding N

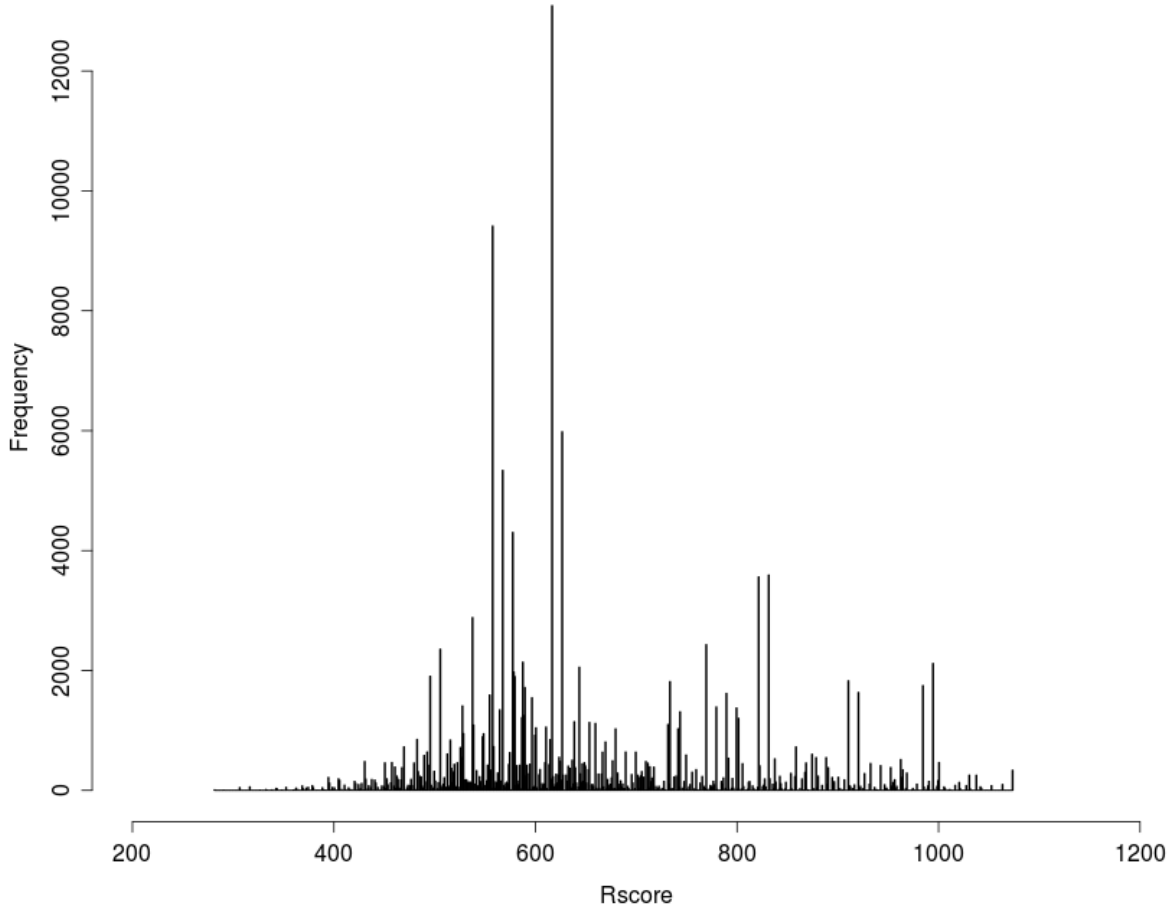
- Nscore = B vs N
  - Concentrate on discriminating between the two similar outcomes
  - Development population concentrated at B+N end of the space
    - Scorecard might not be so good at the G end
      - Unimportant because there are few negative outcomes there

Distribution of Rscore



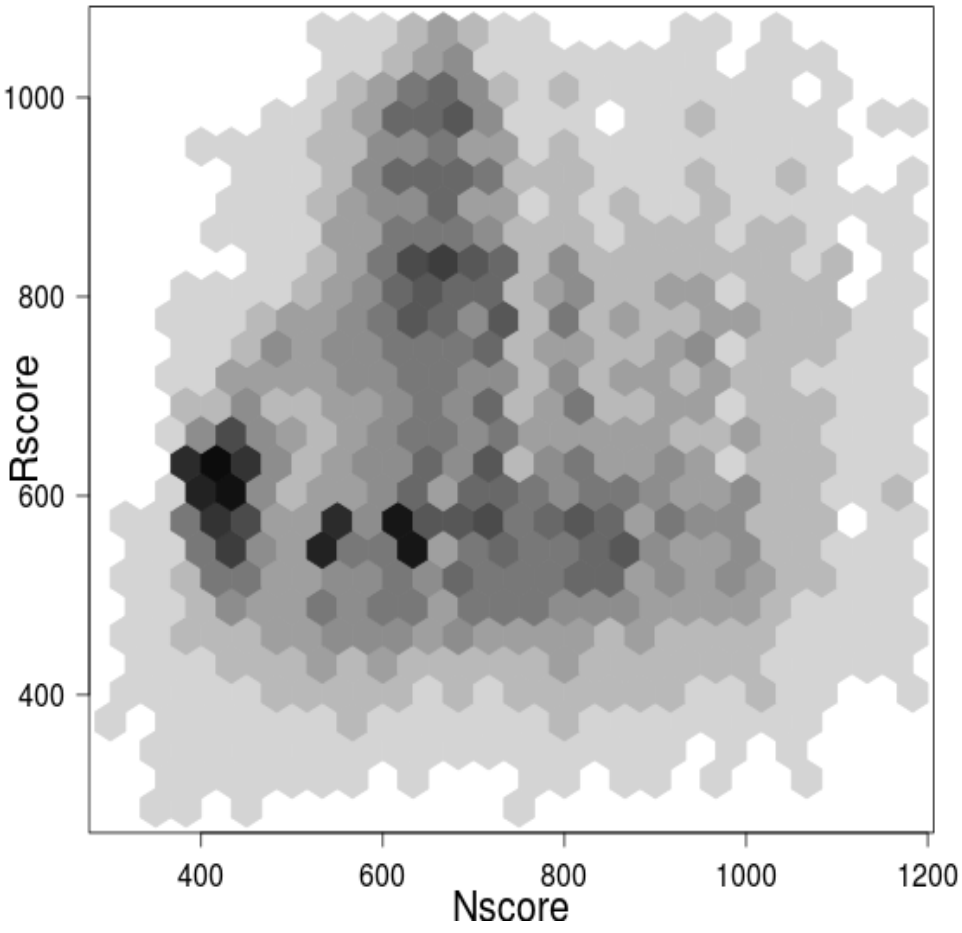
Distribution of Rscore (high res.)

- Typical distribution for a discrete classed scorecard
- Continuous predictors would have been better for joint score use



Joint distribution of Nscore & Rscore

- Good support over joint space
- Some hot spots due to spikes
  - Joint distribution more sparse than marginal
  - Future aim for smoother density distributions
- Density not relevant to Pr(outcome) ~ s(Nscore,Rscore)
- Density relevant to strategy setting (because of lumps)



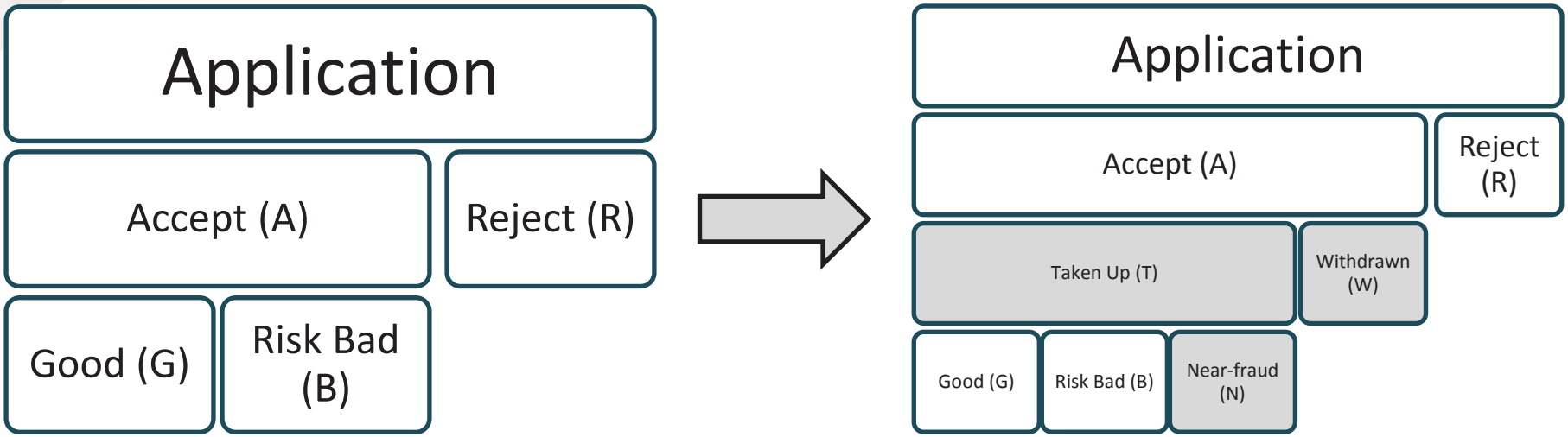
Design issues

Complexity of reject inference

Complexity of reject inference

Problems

- More outcomes than standard reject inference
- Nested outcomes (conditionality)



Solution

- Develop simple conceptual model of reject inference
- Decompose overall RI problem into multiple simpler problems
- Apply conceptual model uniformly across subproblems and recompose

Develop simple conceptual model of RI

Conceptual model of reject inference

Separating the What from the How

- A scorecard is a function: **y = s(p)**
- Scorecard built on all applications is unbiased: **y = s(p; {application})**
- Biased if built on accepts (or other nonrandom subset): **y = s'(p; {accept})**
- In application scoring we have biased **s'( )** and want unbiased **s( )**

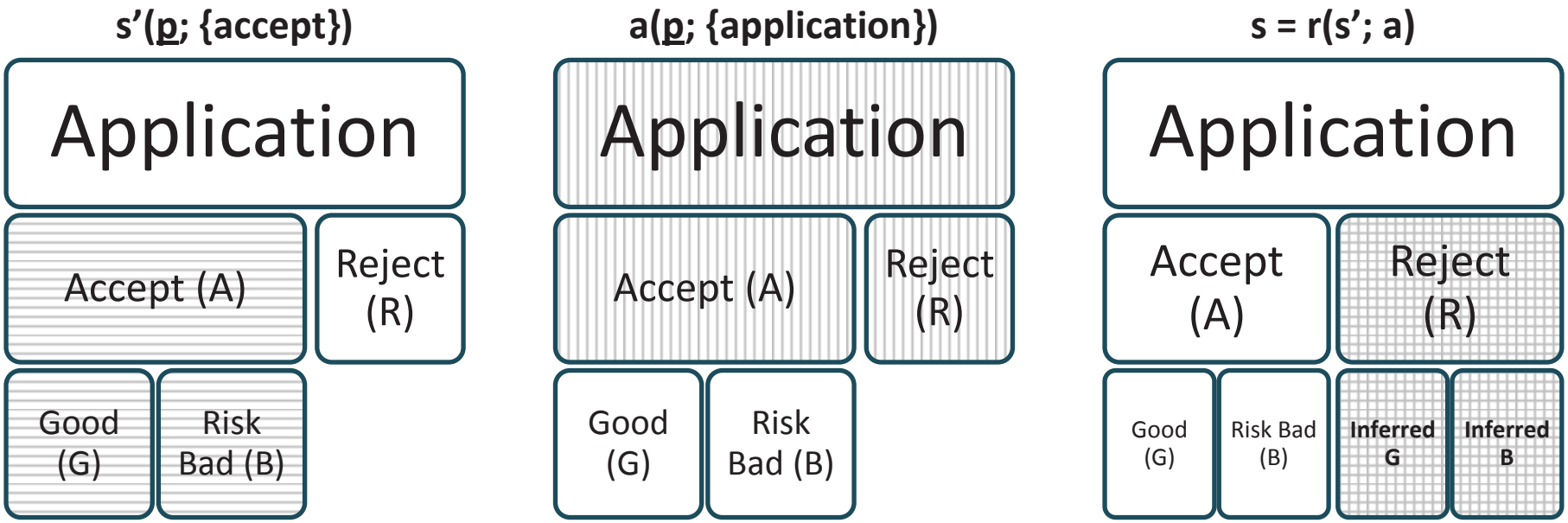
- Think of censoring as a function on scorecard space: **s' = c(s)**
- Difference between **s( )** and **s'( )** depends on strength of censoring process
- Think of **c( )** as modifying **s( )** i.e. **c: s(p) → s'(p)**

- Think of reject inference as inverse of censoring function: $\mathbf{s = r(s') = c^{-1}(s')}$
- Details vary between methods, e.g.: $\mathbf{r(s'; \{a_i(p \cup e; \{application\}) \})}$
  where $\mathbf{a_i}$ are auxiliary models and **e** are extra predictors
- Forgetting the detail, RI is just some modification of a scorecard:

$$\mathbf{r: s'(p) \rightarrow s(p)}$$

Abstracted example of reject inference

- Build model of G/B outcome on accepted cases
- Build model of censoring process (A/R outcome on applications)
- Modify the G/B model on the basis of the A/R model (somehow)
- Apply the modified G/B model to the rejected cases

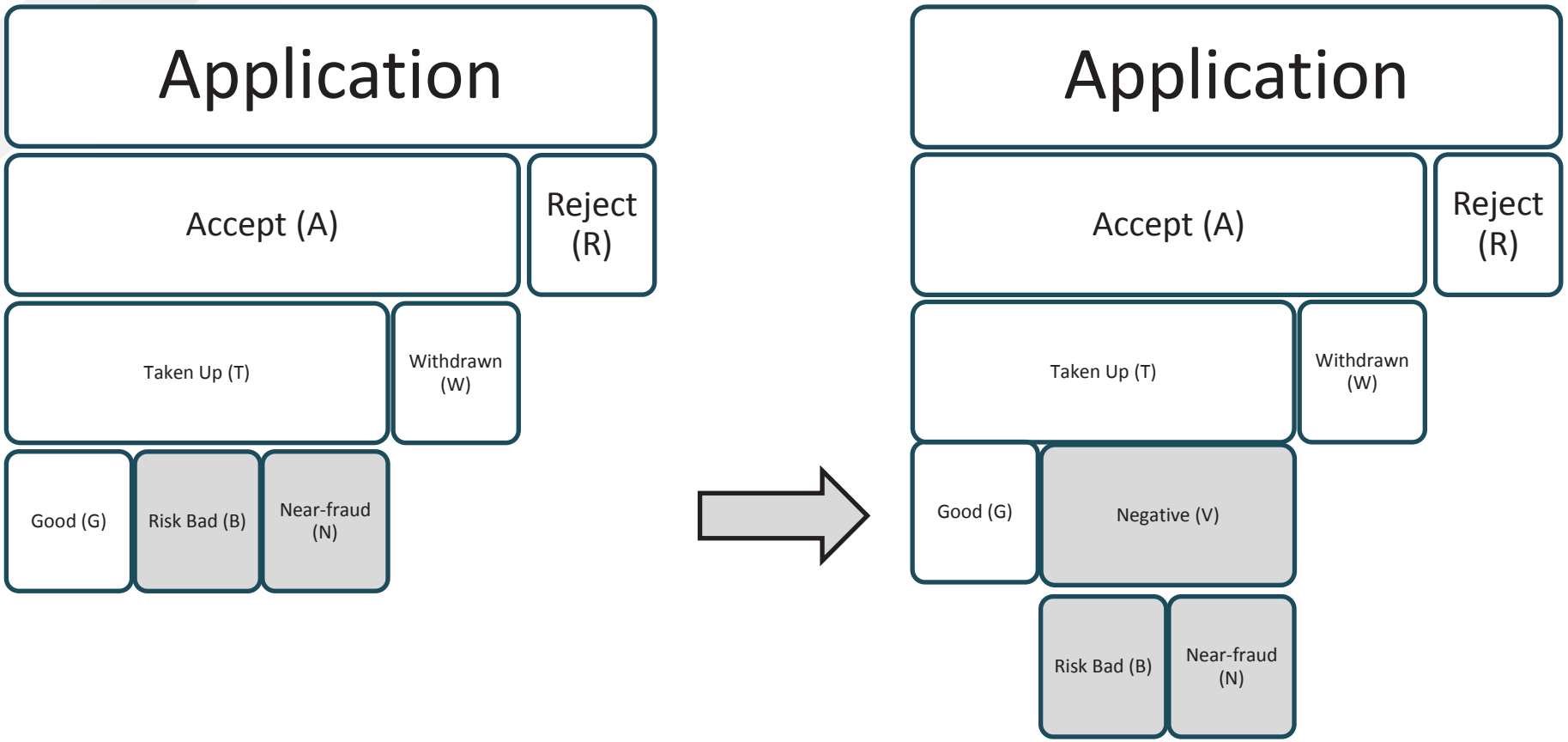


veda
applied intelligence

Decompose overall RI problem into multiple simpler problems

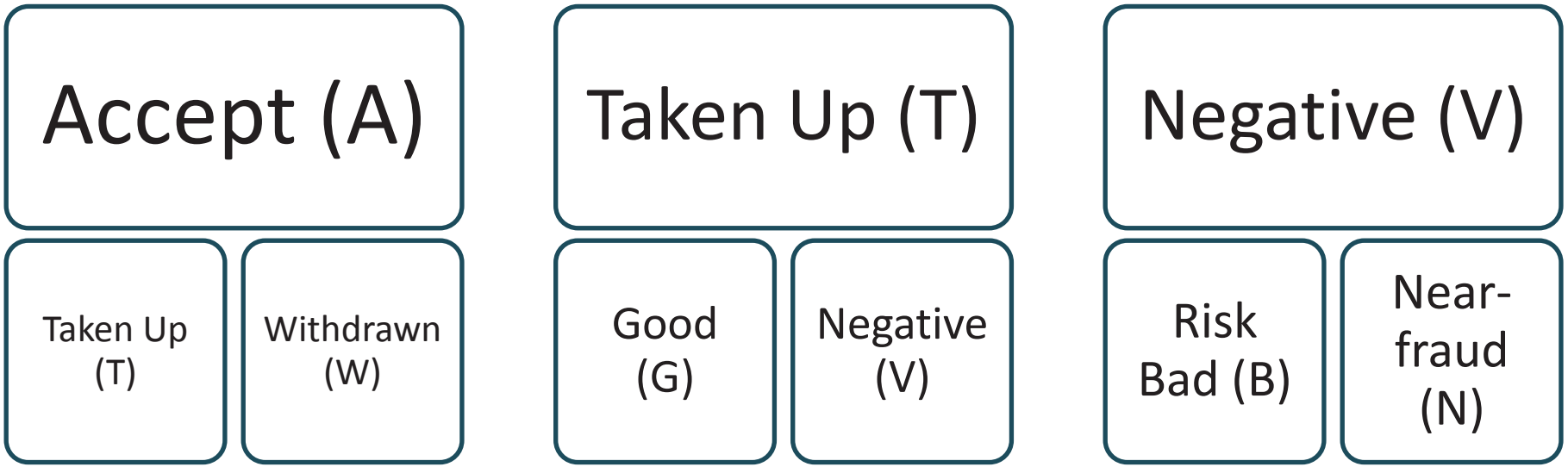
Make the structure more regular

- Make the population flow a binary tree (so each node looks the same)



Decompose the structure

- Break it down into separate reject inference problems
- Build a biased model for each censored population
  - Logistic regression on the cases in each parent node with the immediate children defining the outcomes
  - Predictions are conditional probabilities

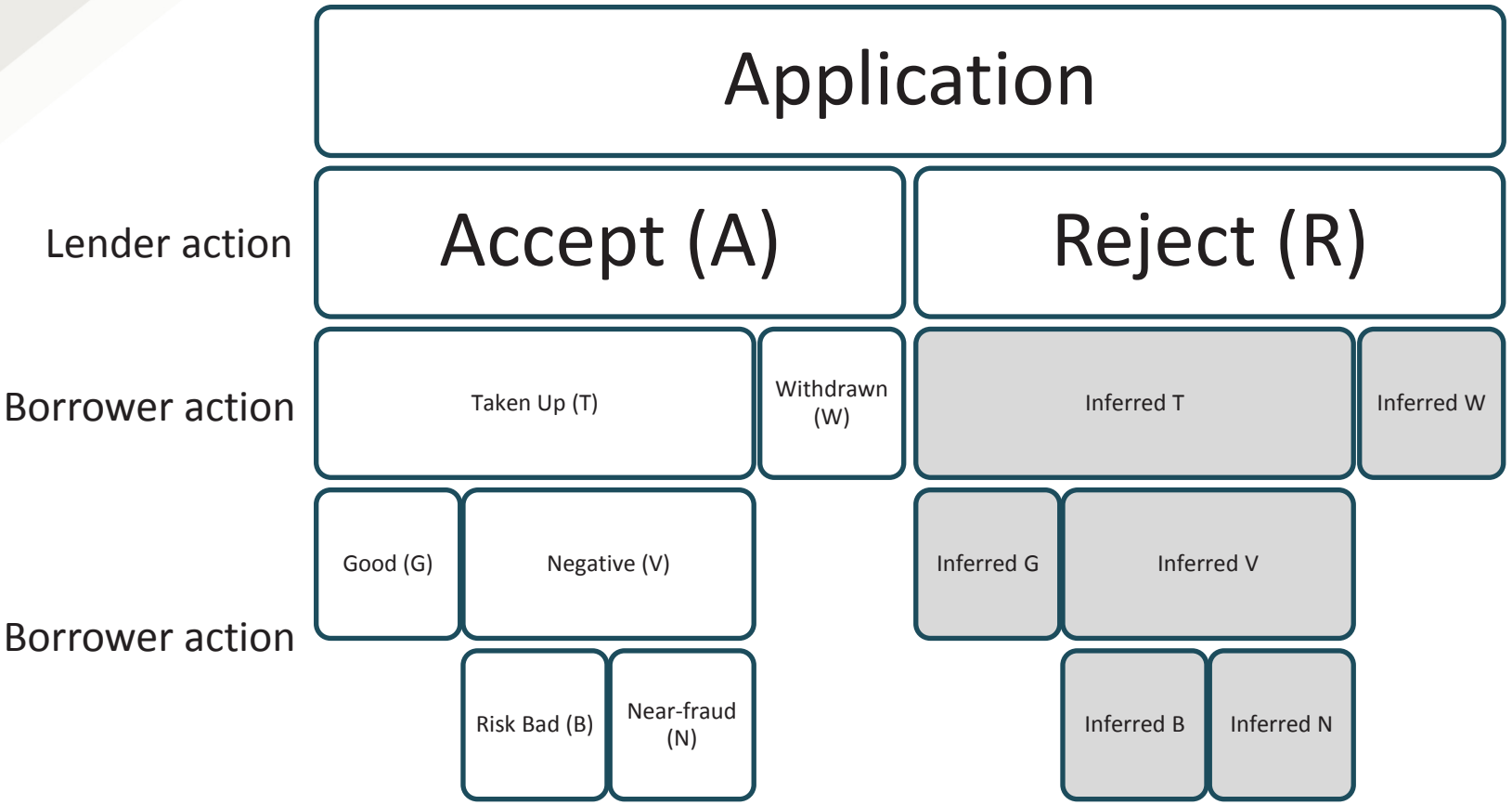


veda
applied intelligence

Apply the conceptual model uniformly across subproblems

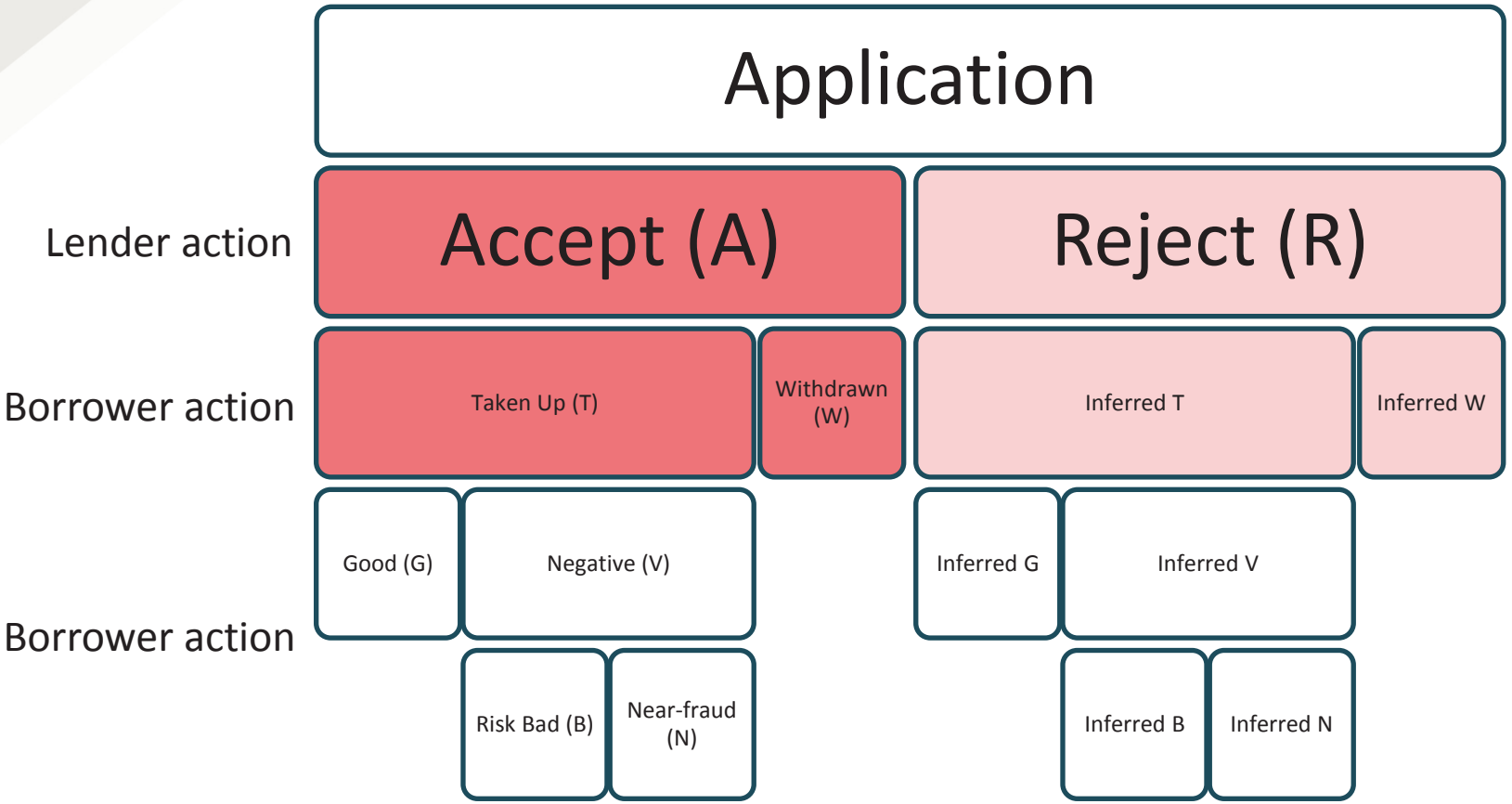
Desired end state of reject inference

- Need to generate inferred probabilities of all outcomes for rejects



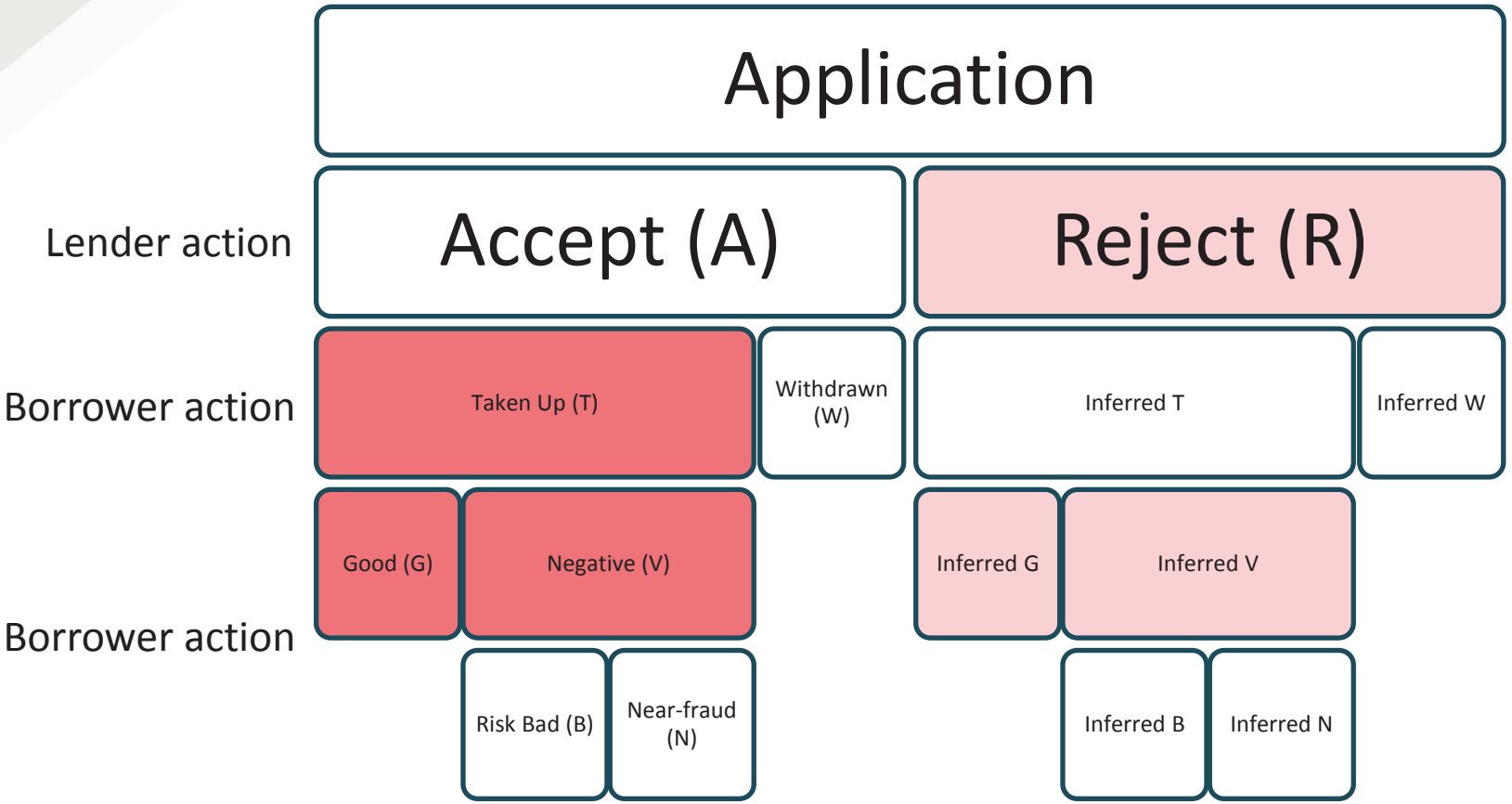
Reject inference
T/W model

- Build T/W model on observed outcomes for Accepts
- Modify T/W model and apply to Rejects



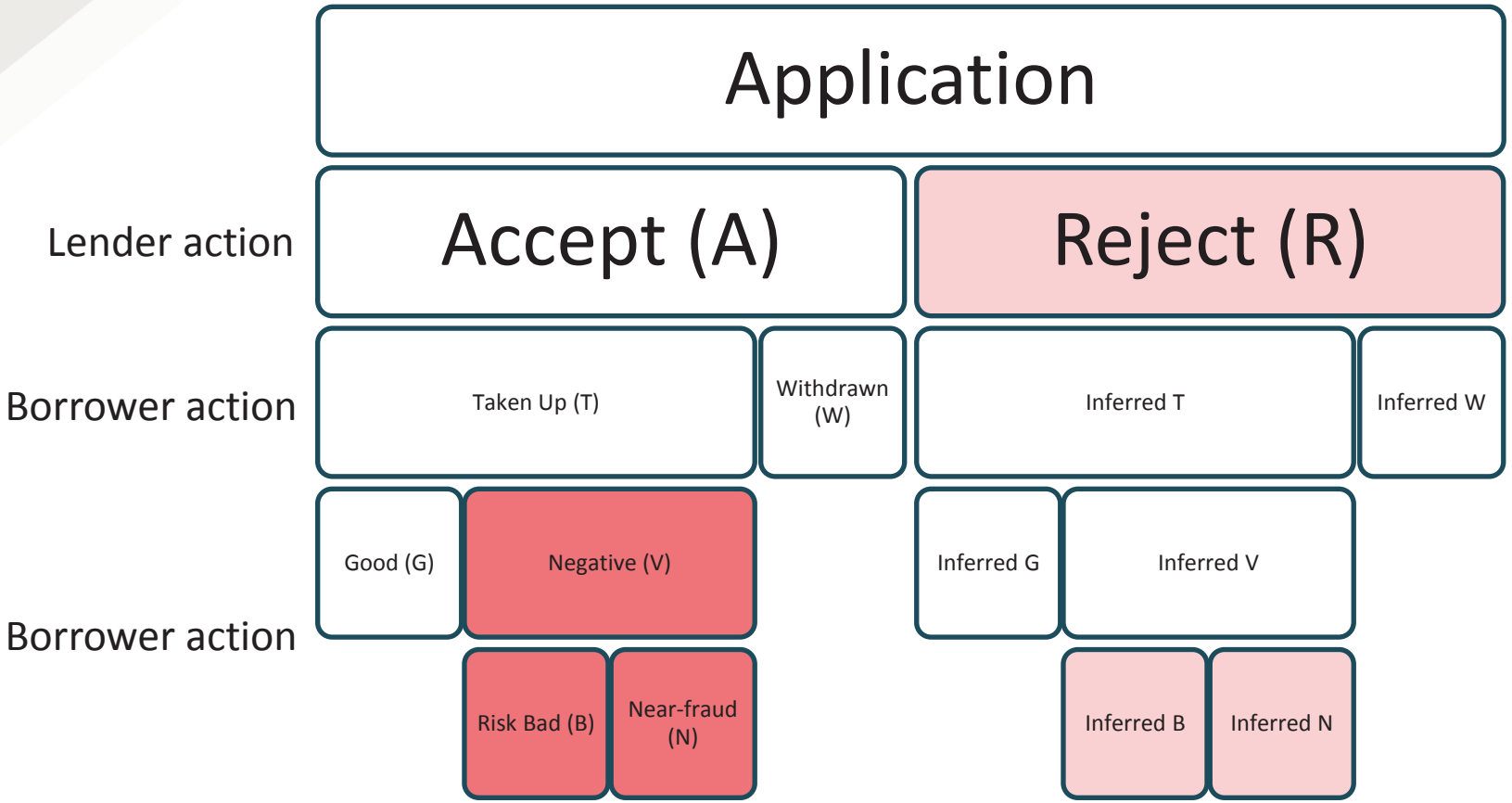
Reject inference
G/V model

- Build G/V model on observed outcomes for Taken Up Accepts
- Modify G/V model and apply to Rejects



Reject inference
B/N model

- Build B/N model on observed outcomes for Negative, Taken Up Accepts
- Modify B/N model and apply to Rejects



veda
applied intelligence

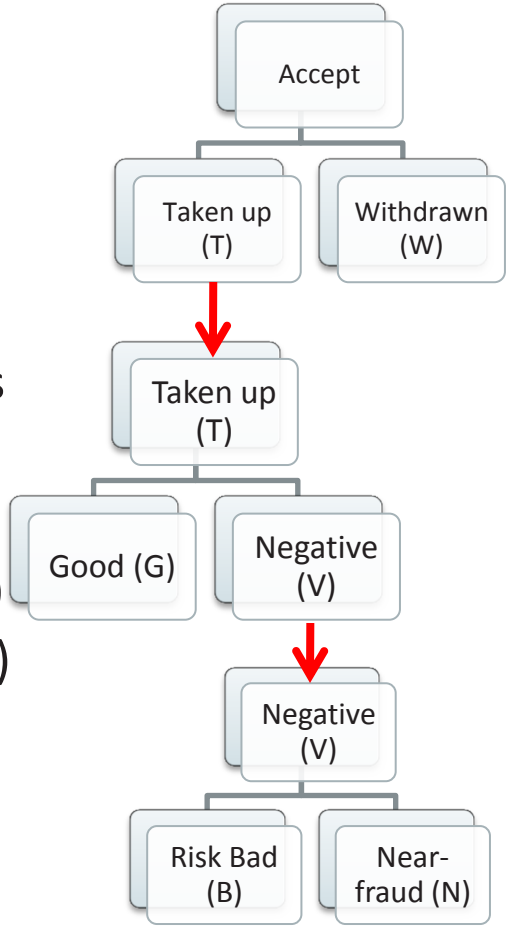
Combining multiple nested models

Combining conditional models

- The modified models predict conditional probabilities:
  - Pr(Taken up | Accept)
  - Pr(Negative | Taken up)
  - Pr(Bad | Negative)
- Every reject has same set of predicted probabilities
- Use logical relationships to calculate other probabilities

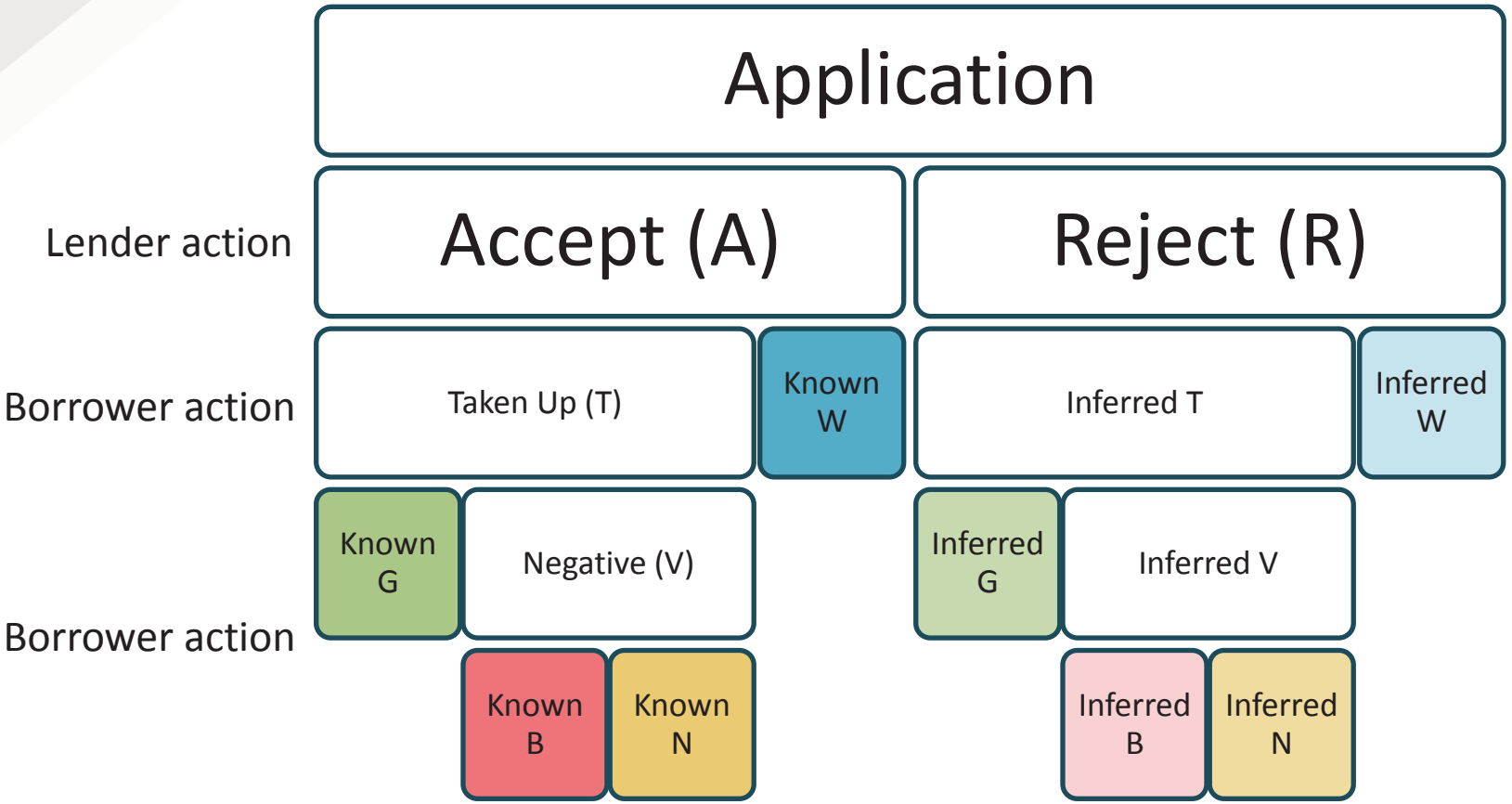
- Calculate unconditional probabilities
  - Pr(child | parent) = Pr(child & parent) / Pr(parent)
  - Pr(child & parent) = Pr(child | parent) * Pr(parent)
  - Pr(child) = Pr(child | parent) * Pr(parent)

- Propagate probabilities down the outcome tree
- Applies to all rejected cases



Unconditional probabilities

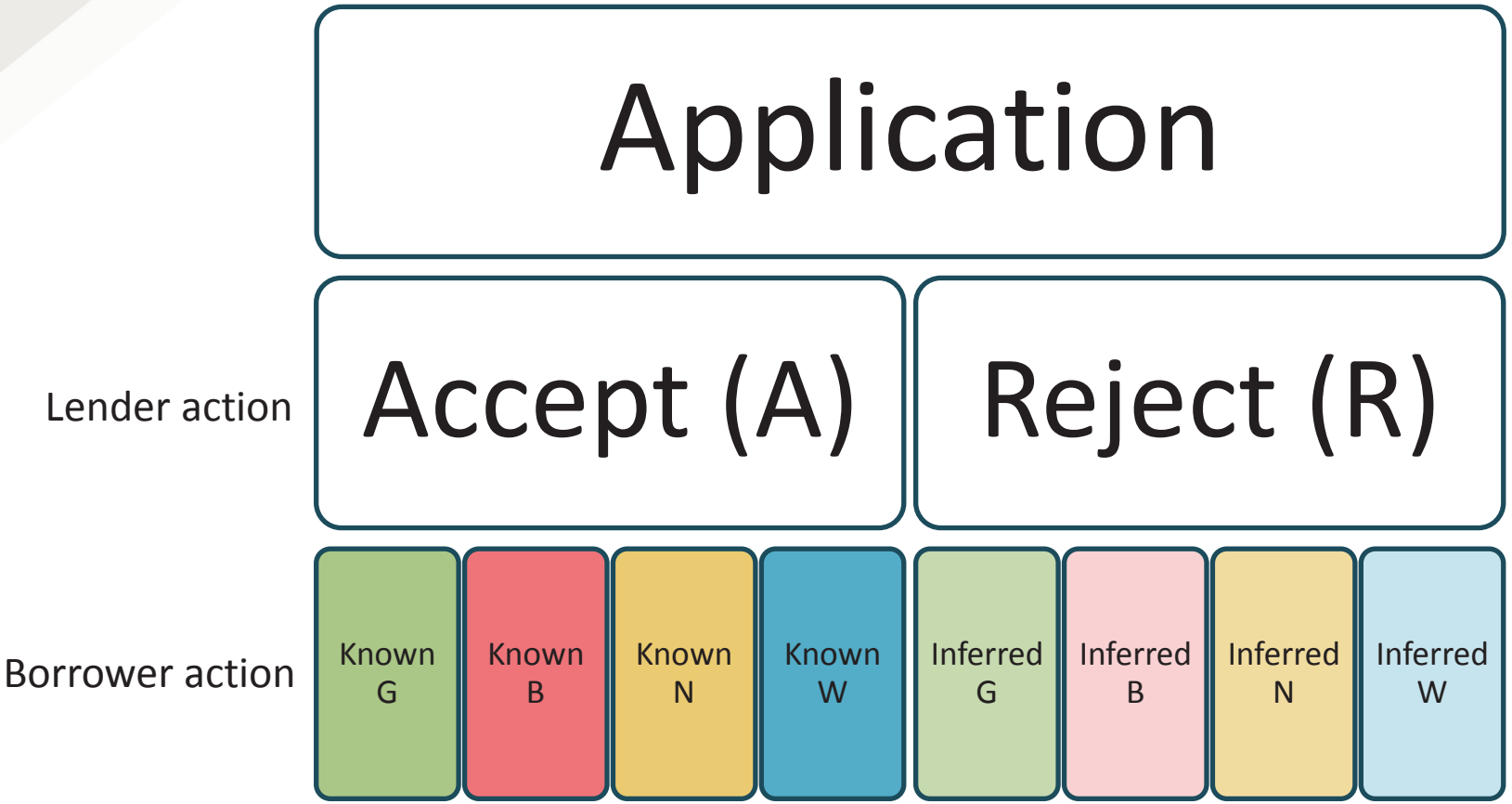
- Probabilities of all outcomes predicted for rejected cases



Realising the inferred outcomes

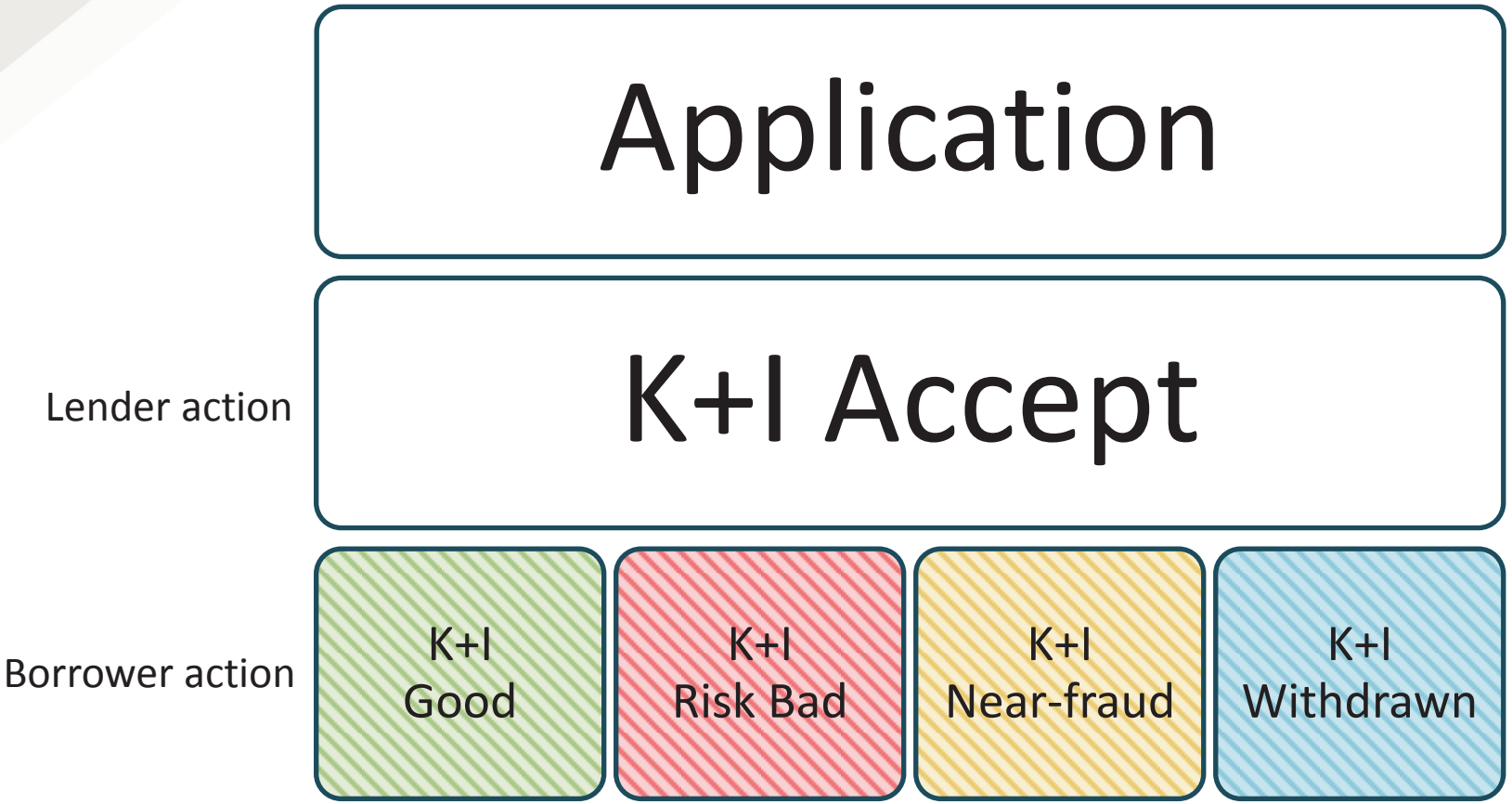
Randomly realise inferred outcomes

- Randomly realise inferred outcomes according to predicted probabilities



Pool known and inferred outcomes

- Pool the corresponding known and inferred outcome cases



veda
applied intelligence

Design issues

More models needed than budgeted

Modelling strategy

Problem

- At least 3 models required for reject inference process
  - Budget for only 2 full model builds (from raw predictors)

Solutions

- Two full models built: Risk (Rscore) and Fraud (Nscore)
  - Standard model builds with standard reject inference for Rscore
  - Guarantees existence of these models as fall-back options

- Use Rscore and Nscore as the only predictors in the nested models
  - Equivalent to dimensional reduction of predictor space

- Use inexpensive modelling process based on cross-calibration

Design issues

Low cost modelling by cross-calibration

Modelling by cross-calibration

Problem

- Traditionally, scorecards built to predict a specific outcome
- More outcomes than specific models (e.g. Taken up vs Withdrawn)

Solution

- Cross-calibrate scores to different outcomes
- Extend to multivariate calibration (Nscore & Rscore)
- Think of this as dimensional reduction of predictor space (similar to PCA)

Problem

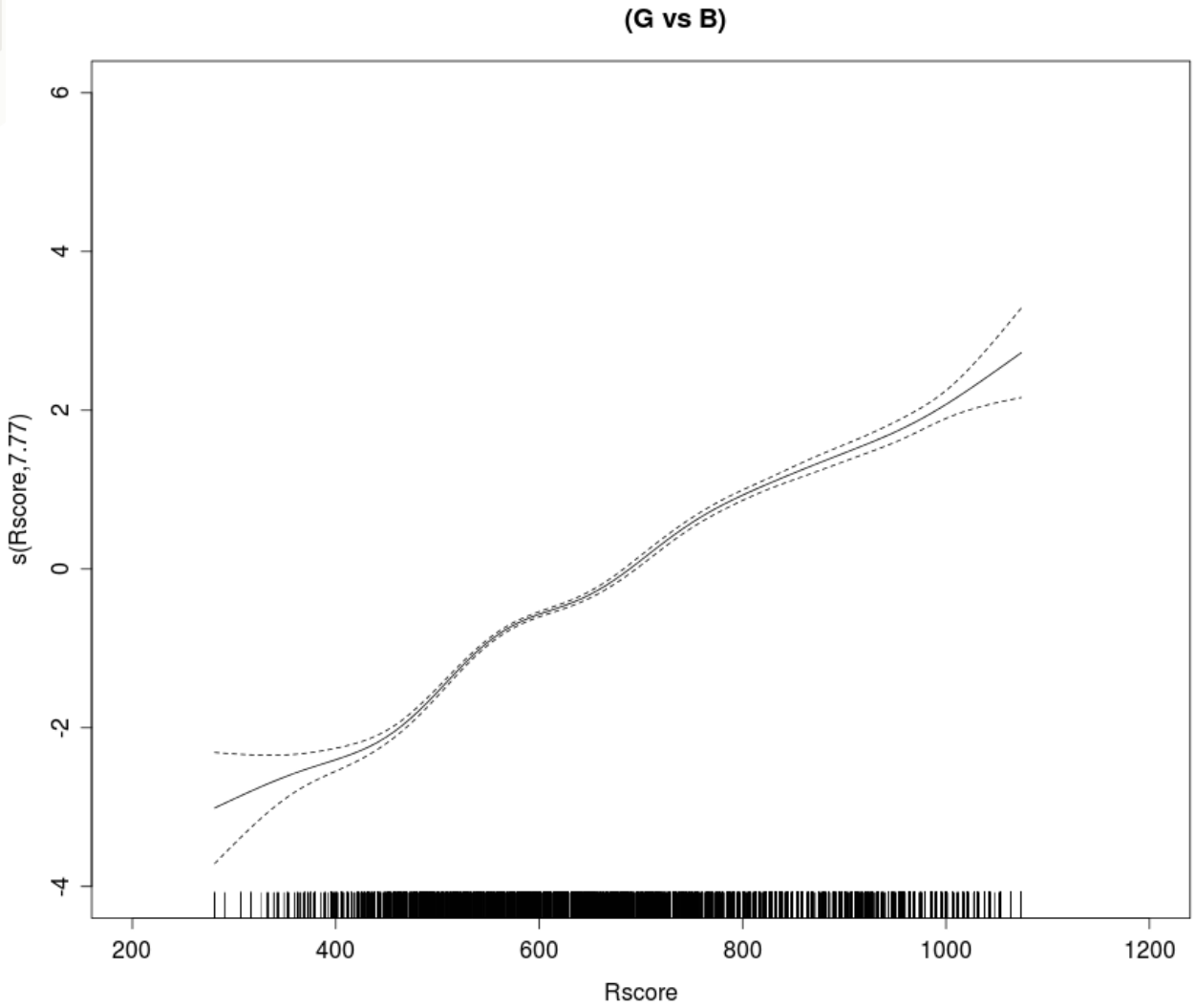
- Traditional methods assume linear response or partition scores into bins
  - Response unlikely to be linear on cross-calibration
  - Binning doesn't make most efficient use of data
  - Binning has problems with sparsity in multivariate extension

Solution

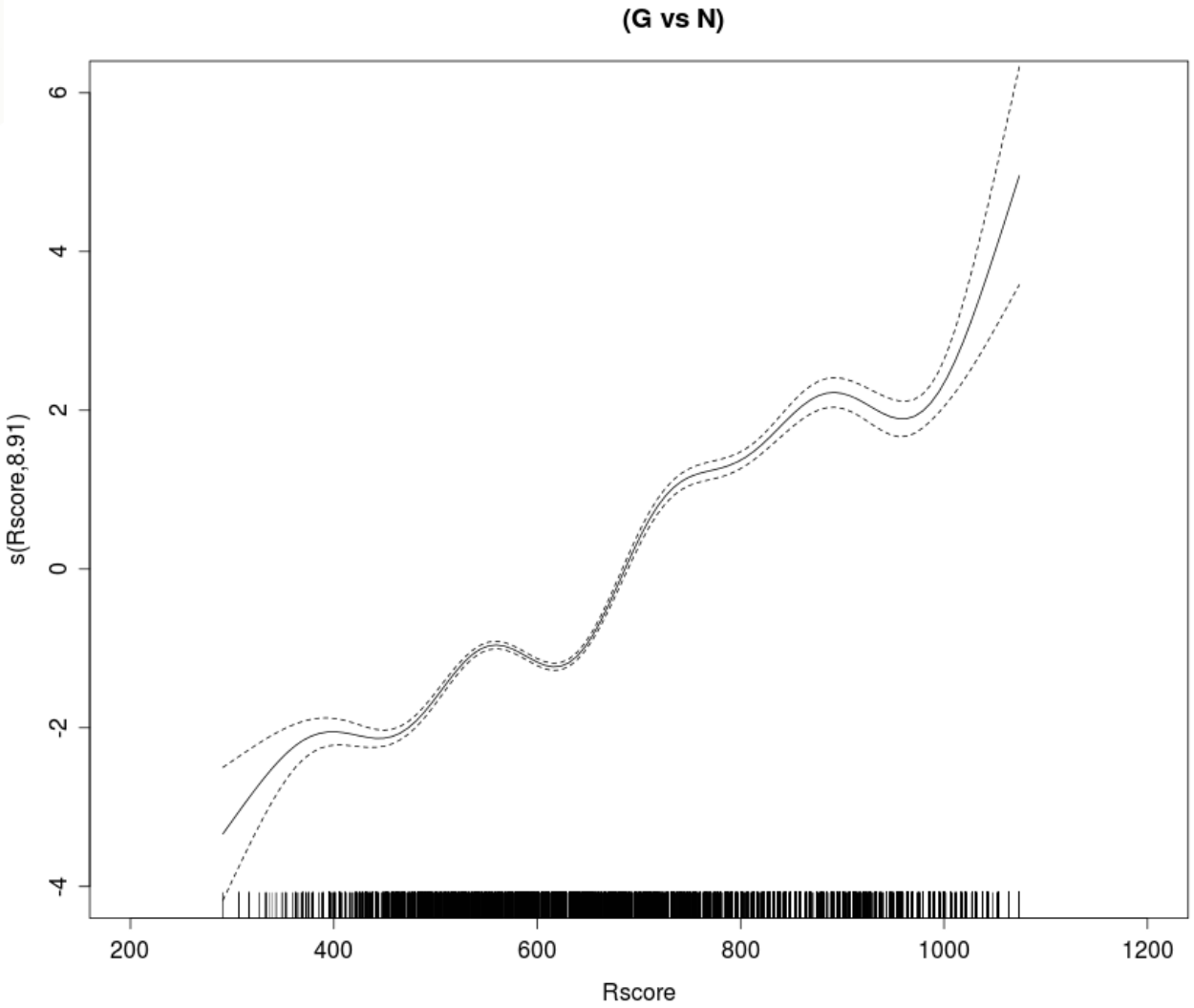
- Use smoothing-spline regression

Rscore calibrations

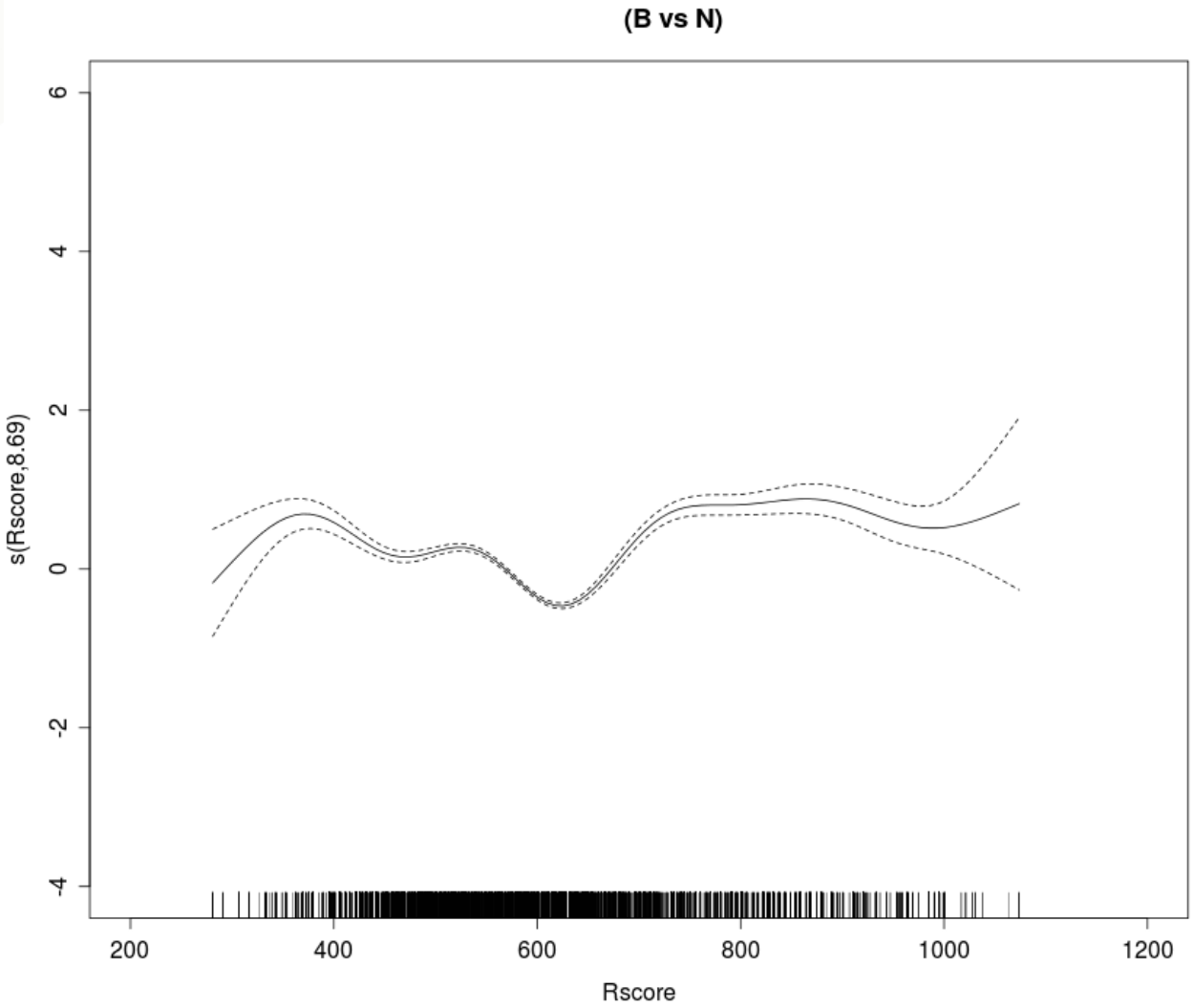
G/B ~ s(Rscore)



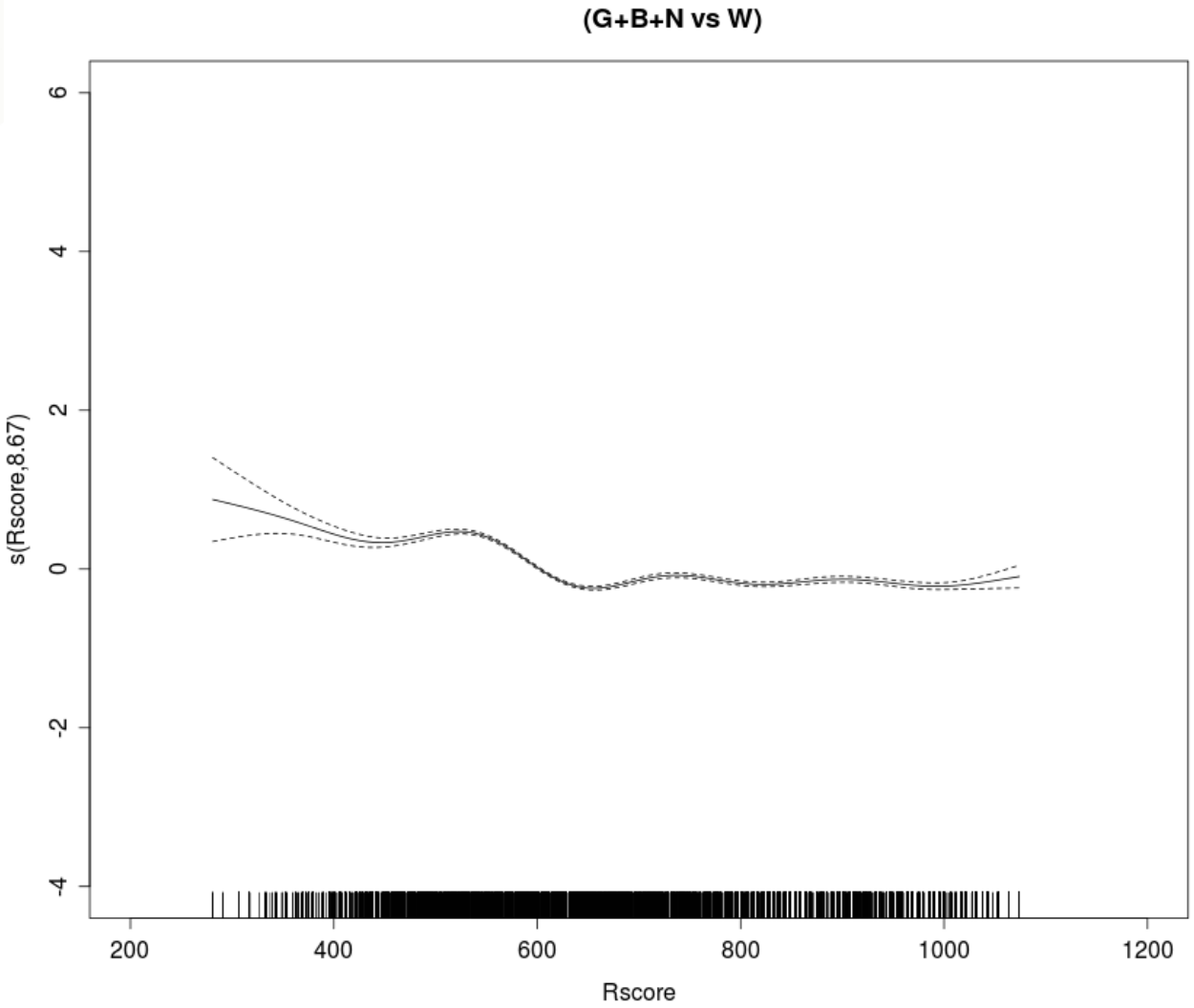
G/N ~ s(Rscore)



B/N ~ s(Rscore)

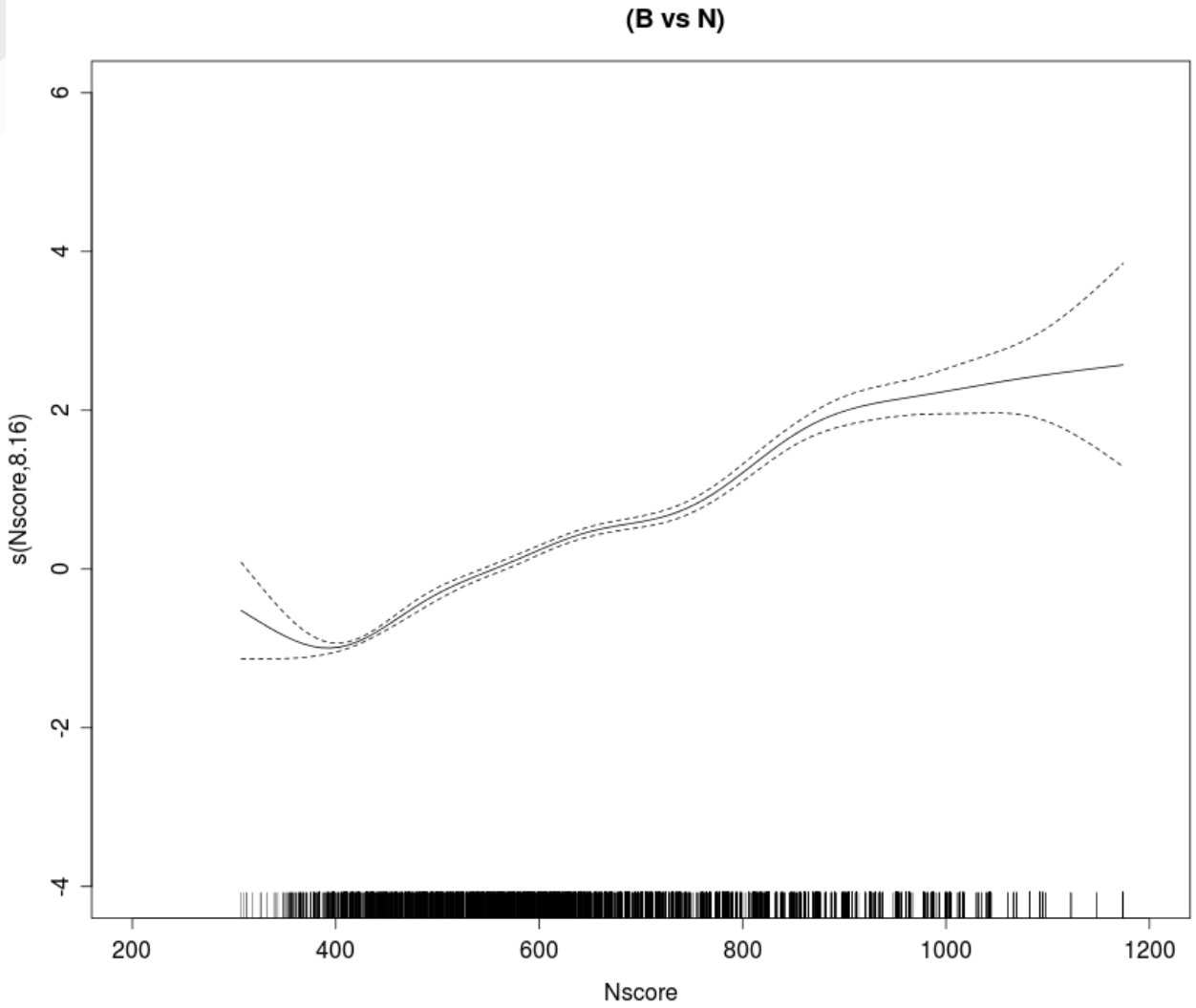


T/W ~ s(Rscore)

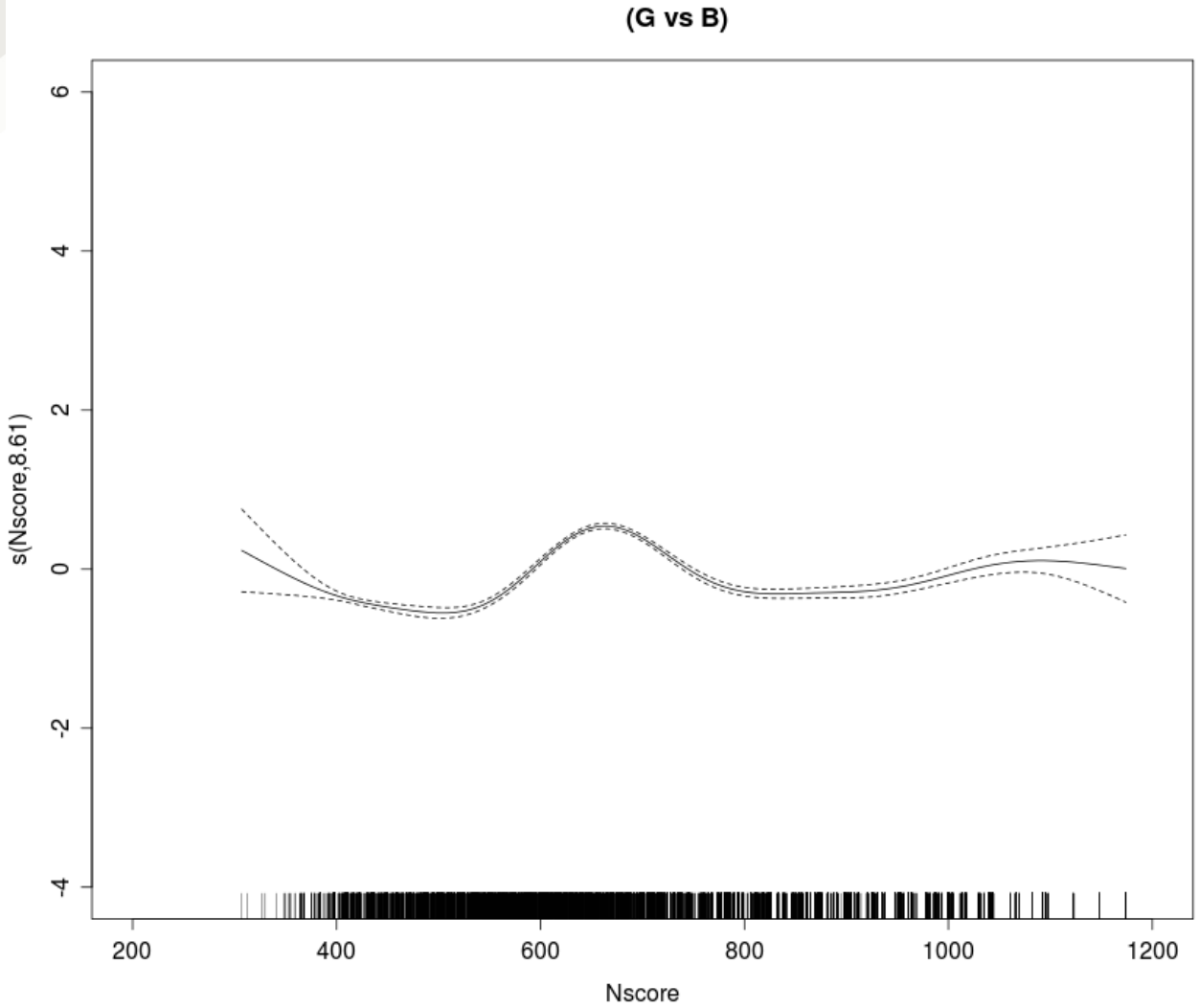


Nscore calibrations

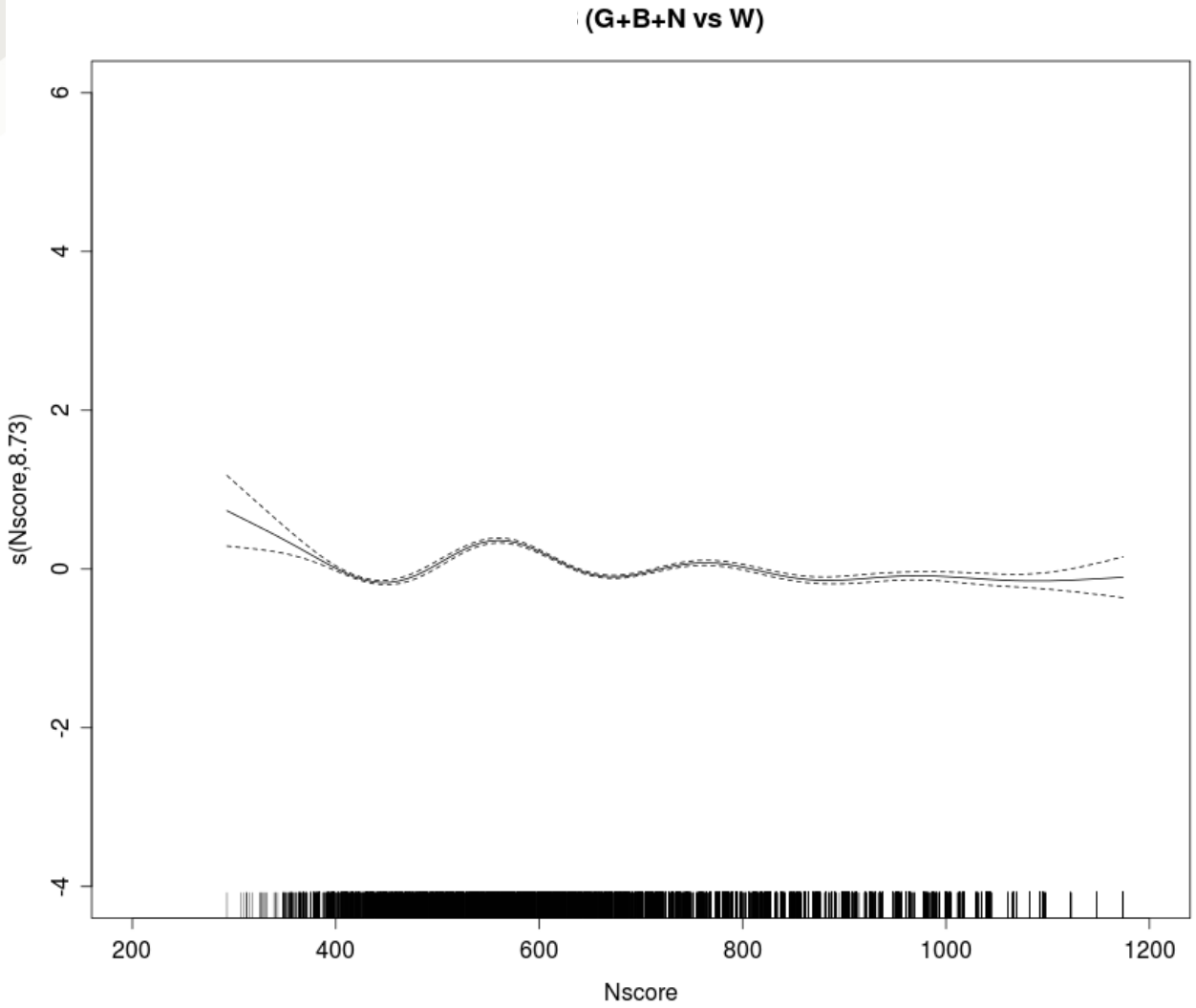
B/N ~ s(Nscore)



G/B ~ s(Nscore)

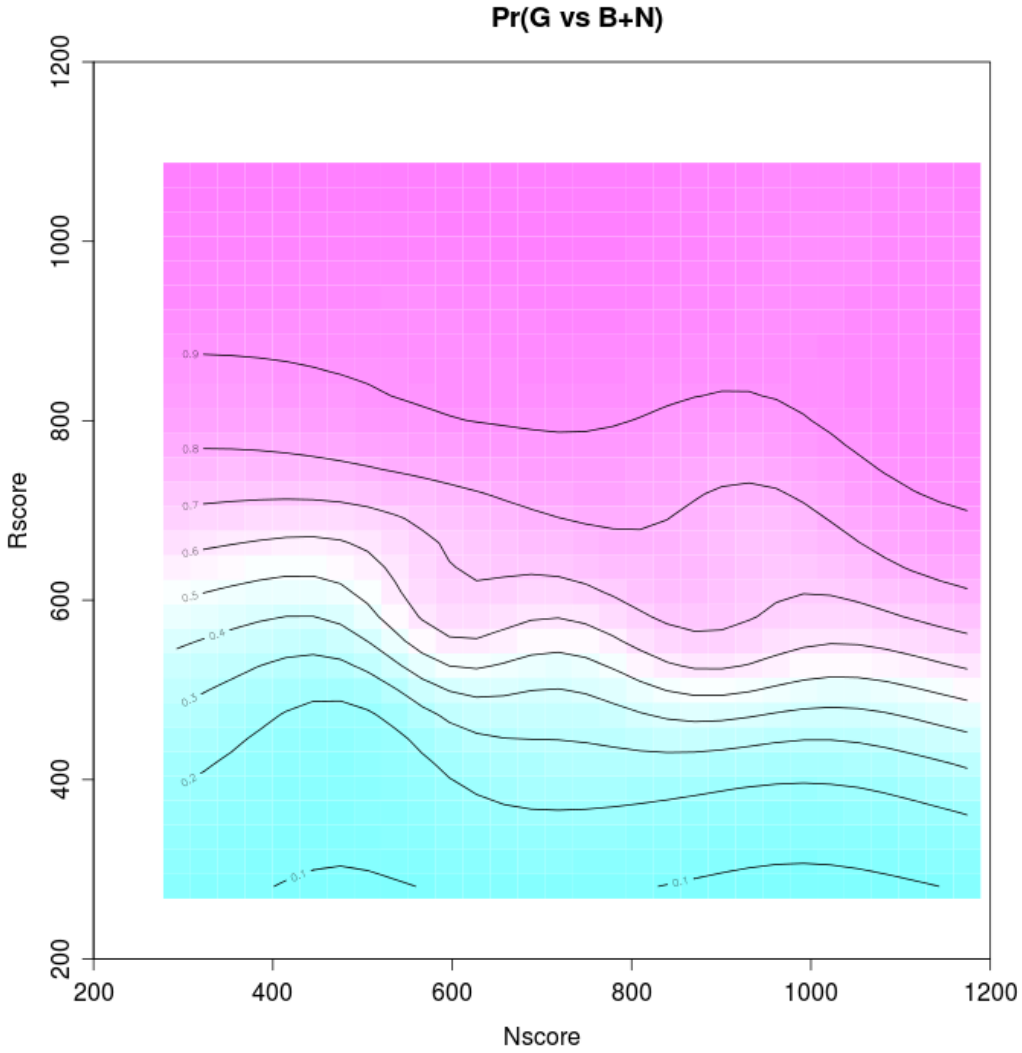


T/W ~ s(Nscore)

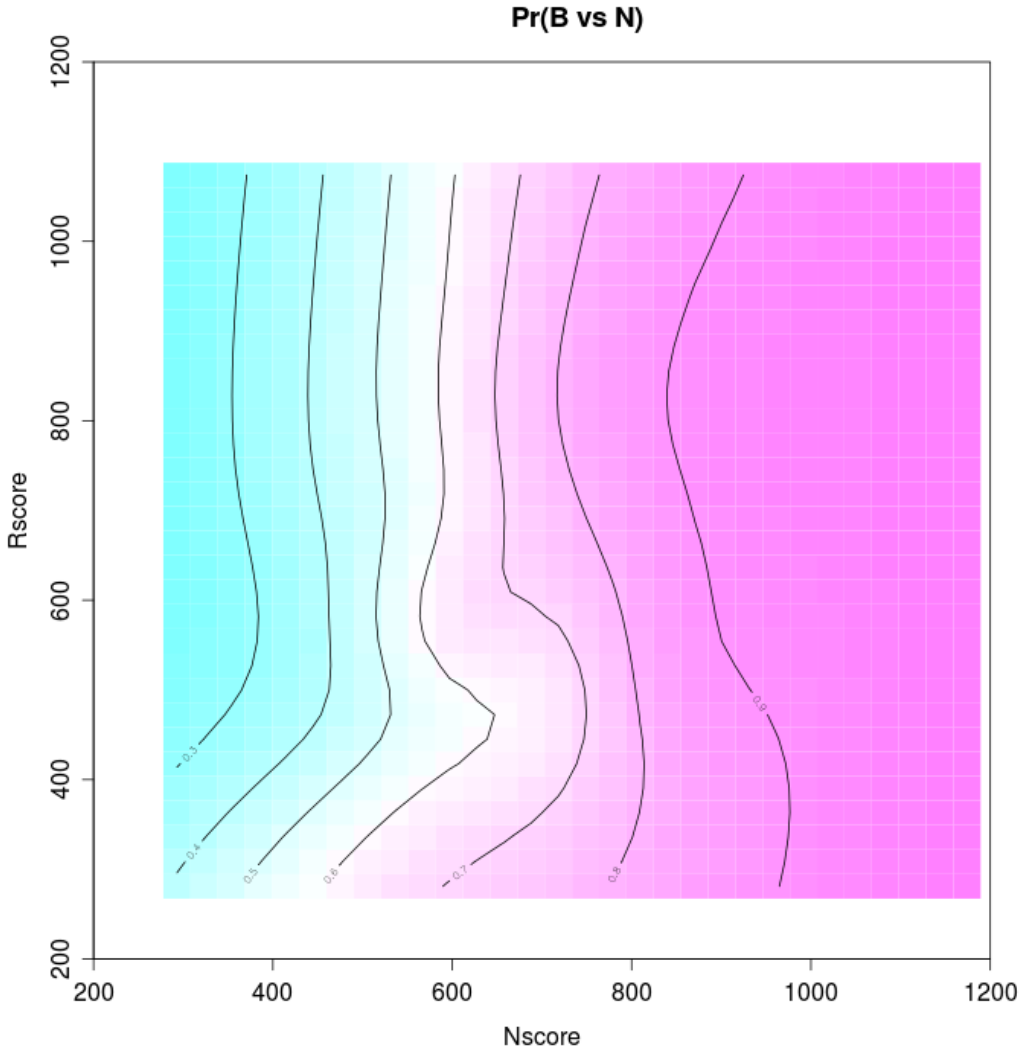


Joint score calibrations

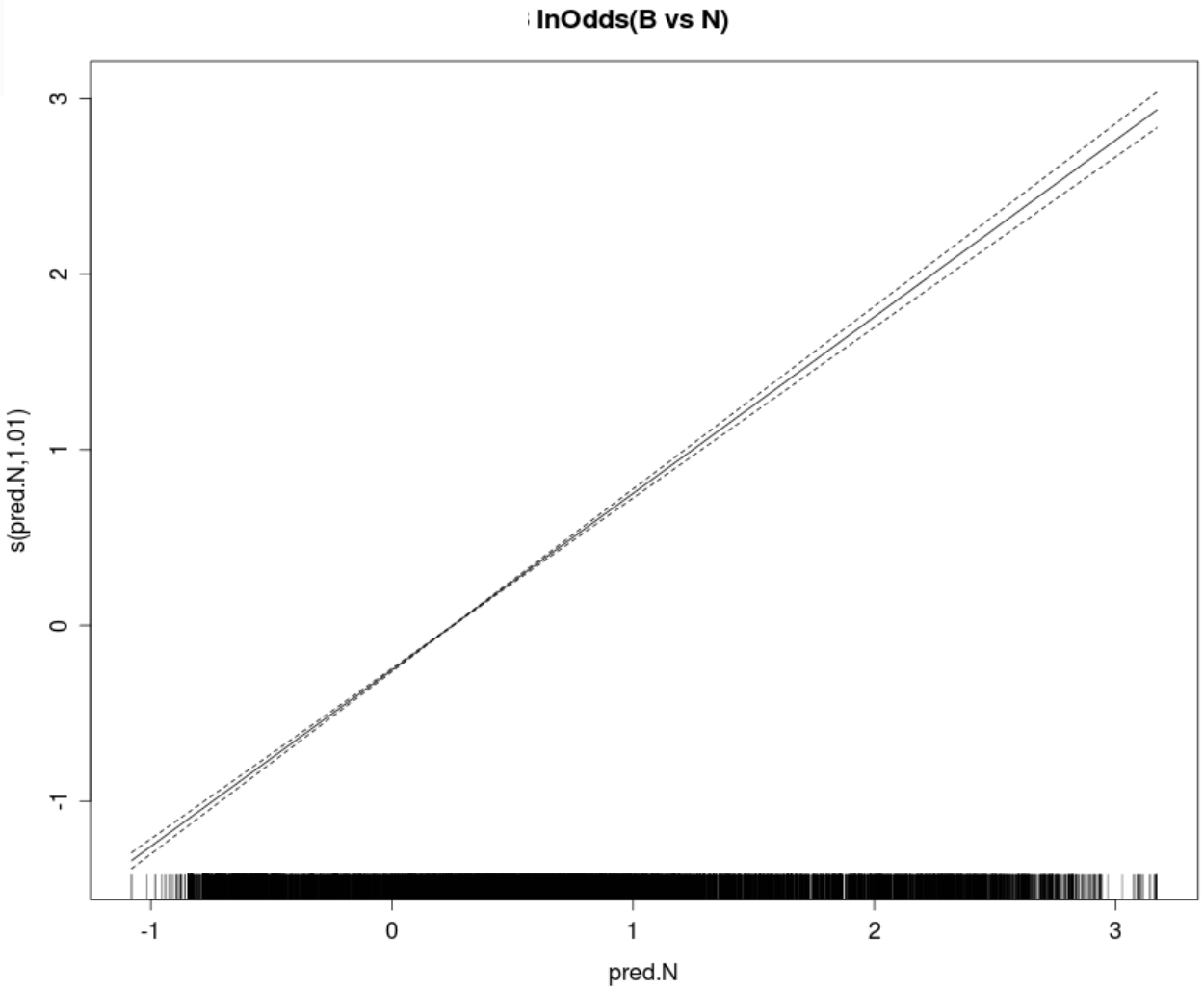
G/V ~ s(Nscore,Rscore)



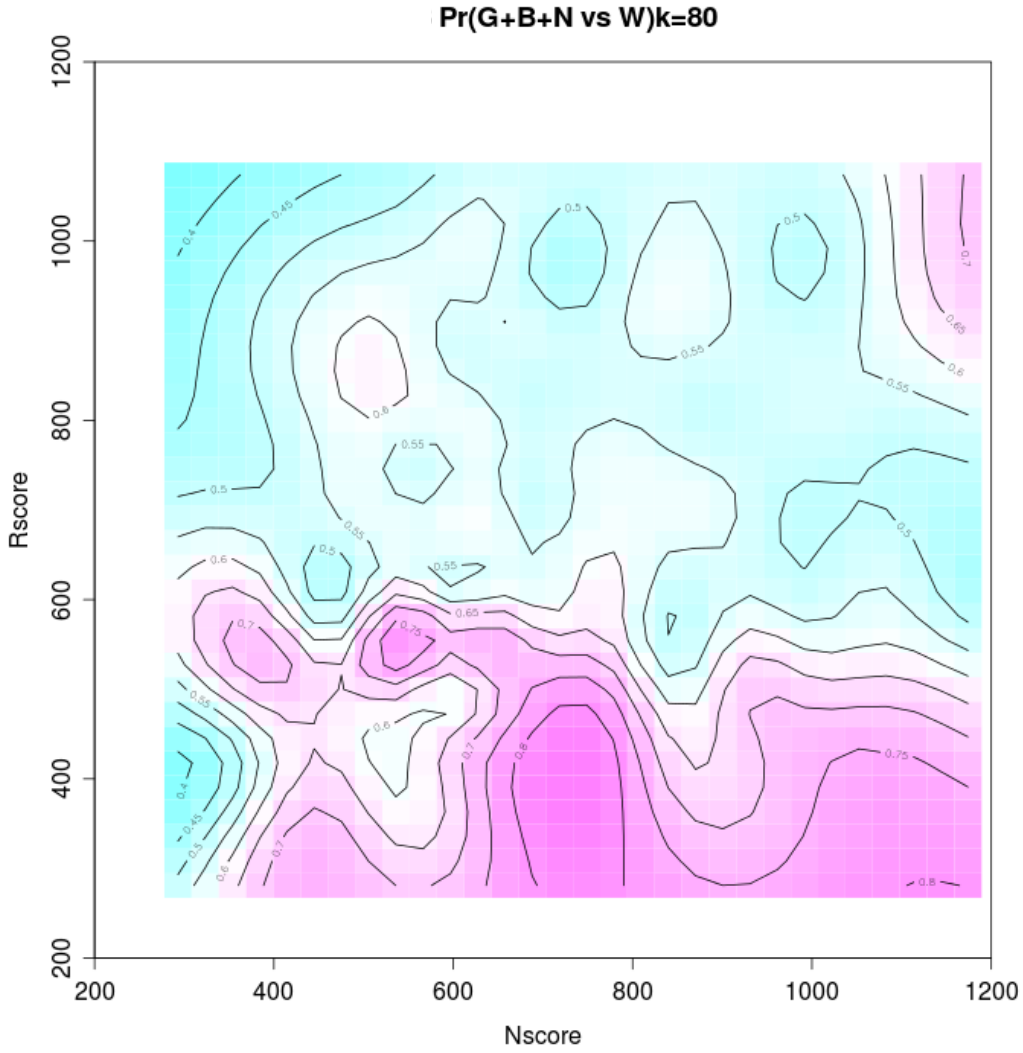
B/N ~ s(Nscore,Rscore)



B/N ~ s(pred(B/N~s(Nscore,Rscore)))



T/W ~ s(Nscore,Rscore; k=80)



veda
applied intelligence

Design issues

Generating usable predictions

Generate predictions for strategies

Problem

- Typical approach is to divide scores into bands
  - Creates a matrix of bins in the joint score space
- Accumulate probabilities of outcomes in each matrix bin
- Binning doesn't make most efficient use of the data
- Binning has problems of sparsity in bivariate matrix
- Coarse bands make fine control of strategy difficult

Solution

- Fit outcomes to known + inferred data with smoothing spline regressions

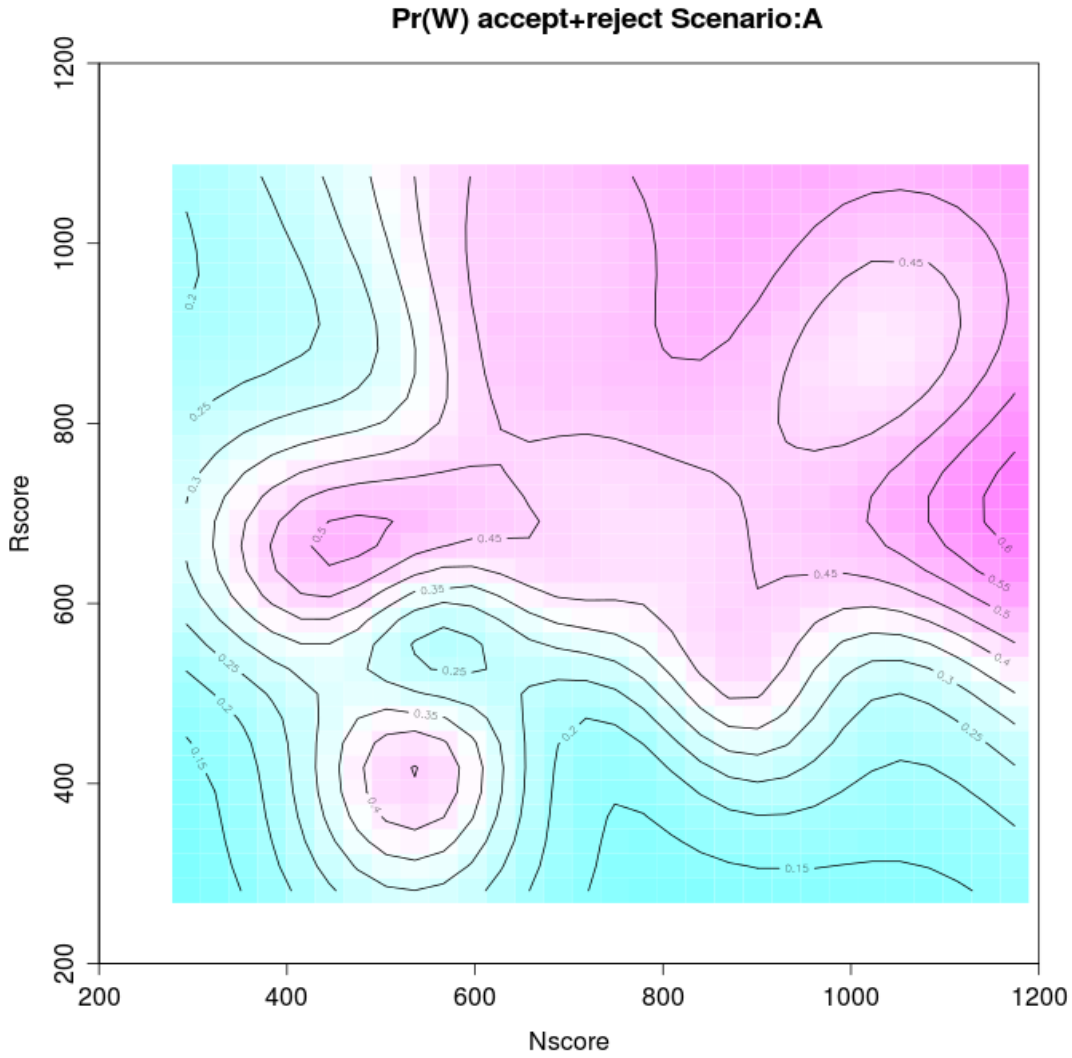
Problem

- User systems don't implement smoothing splines

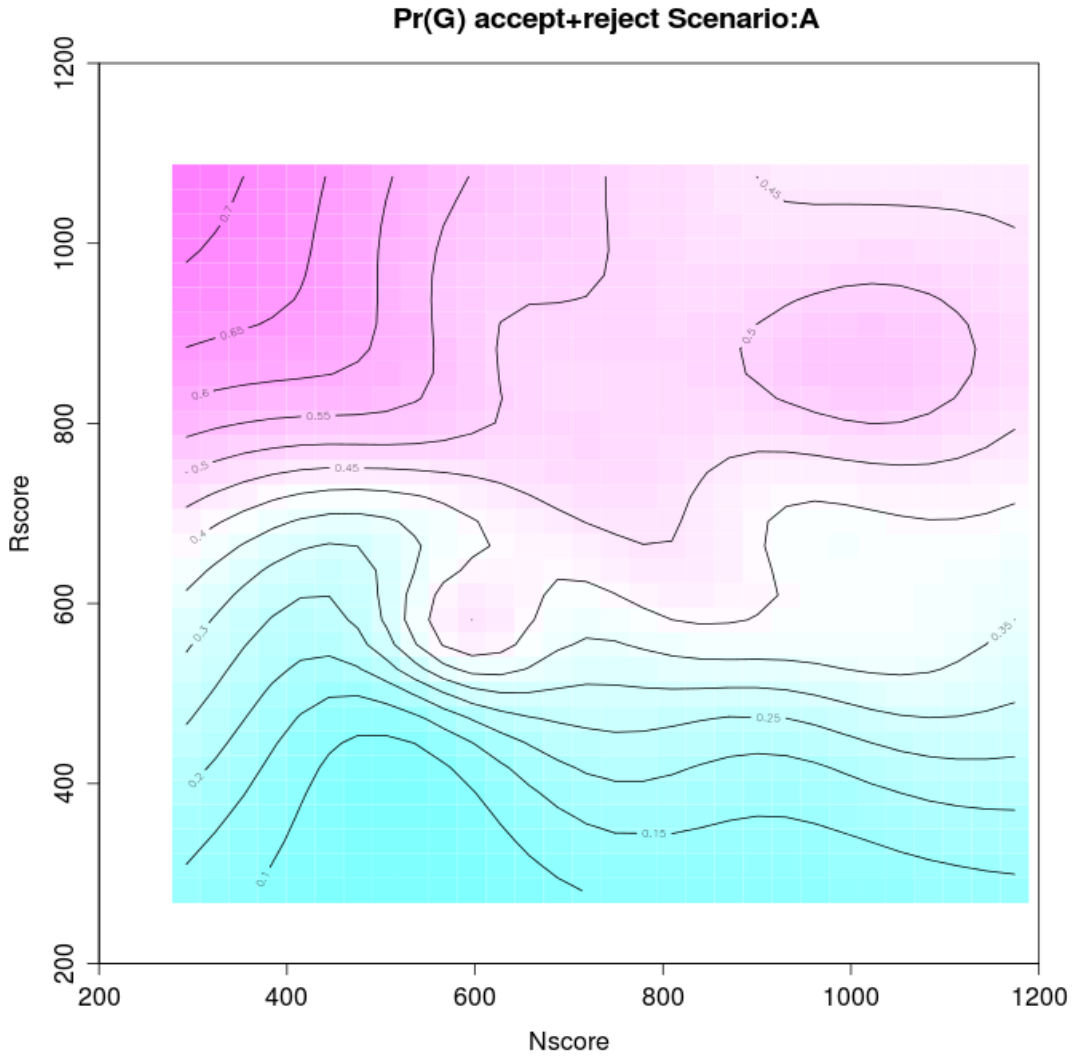
Solution

- Make predictions at a fine grid on the smoothed model
  - Smooth first, make discrete later

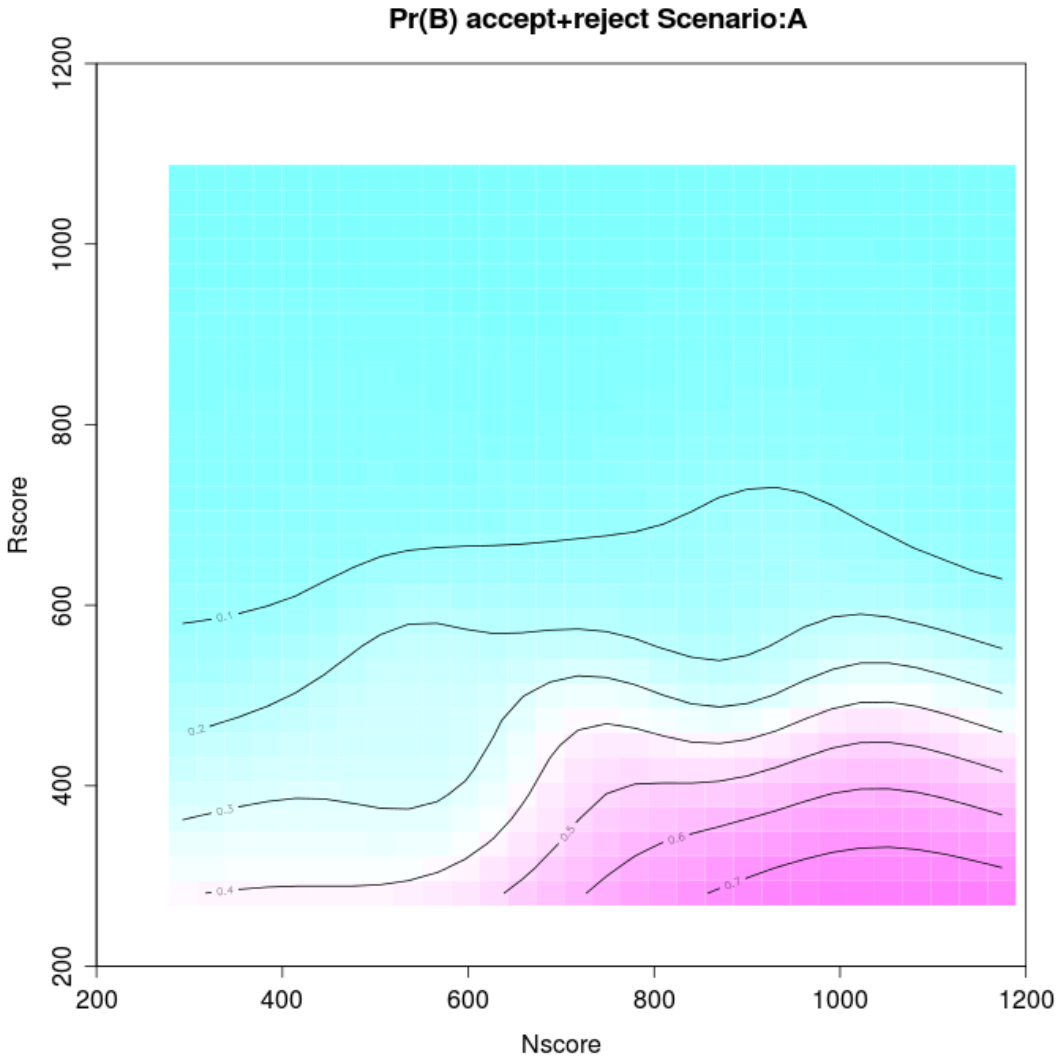
W/(G+B+N) ~ s(Nscore,Rscore)
(known + inferred)



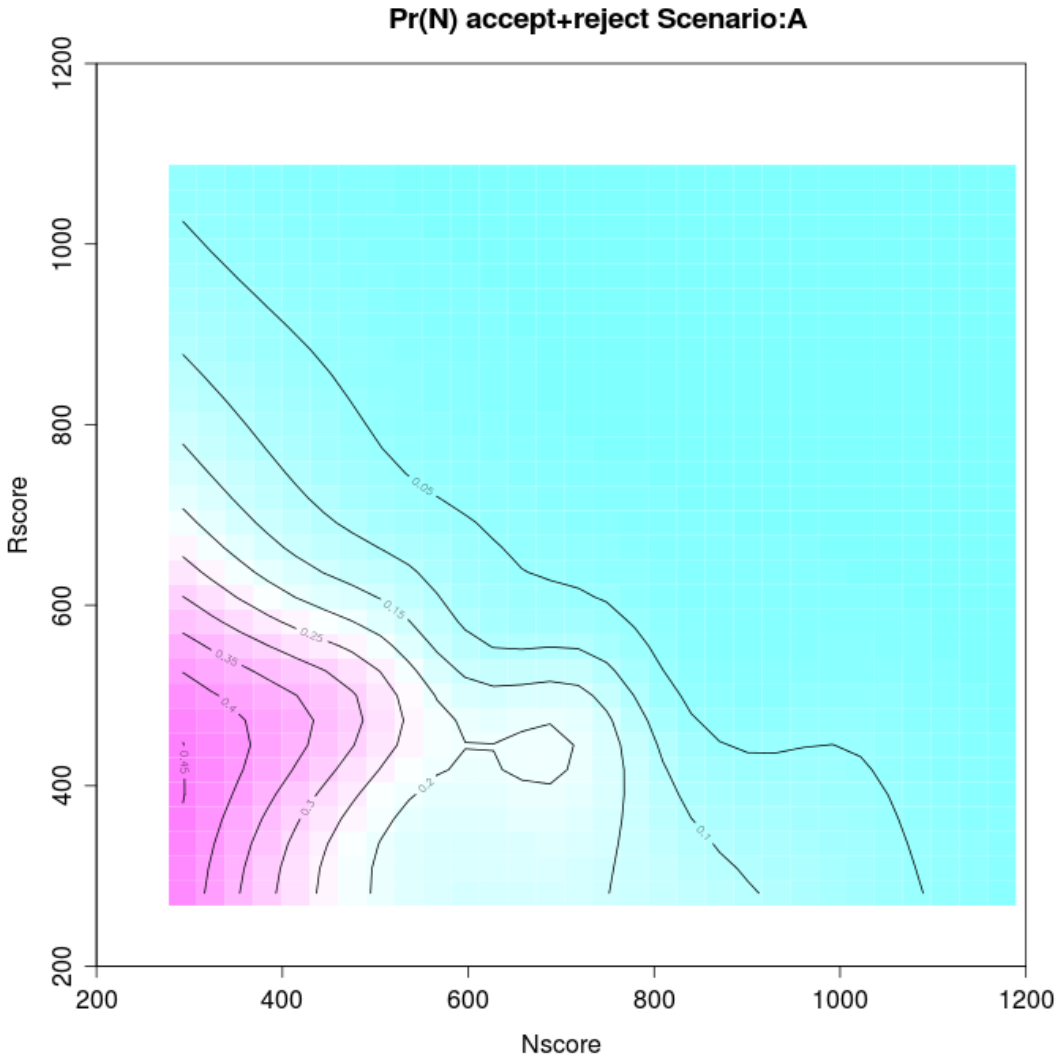
G/(W+B+N) ~ s(Nscore,Rscore)
(known + inferred)



B/(W+G+N) ~ s(Nscore,Rscore)
(known + inferred)



N/(W+G+B) ~ s(Nscore,Rscore)
(known + inferred)



veda
applied intelligence

Questions?