

Preference Inference Using Stan

John Mount
Win Vector LLC
<https://www.win-vector.com/>
September 12, 2024

Win Vector LLC

- Win Vector LLC is a statistics, machine learning, and data science consultancy and training organization.
- Specialize in solution design, technology evaluation, and prototyping.
- Contact: jmount@win-vector.com .

Outline

- What is Stan
- The problem
 - Some history
- Solution
 - Experiments/Results
 - Methods
- Conclusions/recommendations

What is Stan?

Stan

- A Markov Chain Monte Carlo sampler
- Can be used for complicated Bayesian inference
 - Guesses likely values of parameters, and nuisance variables conditioned on observed data.
 - Returns distributional answers, allowing us to see if there incompatible solution.
- A massive piece of software.

The Problem

The Situation

- Users (or user personas) have intrinsic unobserved **preferences** or valuations
 - Personas are either representatives of a group of users or a label collecting together a group of users.
 - This lets us assume we can collect a lot of per-persona data.
- We observe user **behaviors**
 - Typical observation: we presented 5 alternatives and the user purchased one.
 - Observation contaminated both by presentation position and alternative items in the presentation.
- Can we estimate persona preferences from persona behaviors?
 - Often called “learning to rank.”
 - We can use estimated preferences to plan (a lot more than just predicting future list behaviors).

Our Goal

- Estimate the user (or user cohort) *intrinsic* preferences (independent of presentation position and other items).
- Allows us to rate items with respect to user to retrieve, sort, estimate utilities and so on.
- Eliminates some systems, such as `xgb.XGBRanker` (as it reproduces predictions over whole lists)
- From: https://xgboost.readthedocs.io/en/stable/python/examples/learning_to_rank.html#sphx-glr-python-examples-learning-to-rank-py

```
X_test, clicks_test, y_test, qid_test = sort_ltr_samples(  
    test.X,  
    test.y,  
    test.qid,  
    test.click,  
    test.pos,  
)
```

```
...  
ranker.predict(X_test)
```

(notice the `qid` column, which means data is in presentation and we `xgb.XGBRanker` is compute probability of selection with respect to the alternatives, not utility. Would have to “sum out” many proposed panels to get intrinsic utility.)

Our Thesis

- Factor overall system performance into the product of:
 - “**Model Structure**” (how well the explanatory variables can reproduce scores).
 - We used logistic regression (great at converting yes/no observations back into coefficients and probability scores).
 - “**Presentation Un-Censoring Hygiene**” (how well we account for the censoring effects of presentation on the problem). We are treating this as a *required user supplied causal structure*.
 - May involve joining in some “facts about presentation” and “user profile” interactions.
- To improve: must know where we *actually* are losing quality.
 - Stan can simulate inverting presentation censorship.

Example Problems

- User is shown 5 products online and clicks on one.
- User tastes 5 wines and buys at most one.
 - (repeat with 100 users thought to be in same persona)



We **wish** they would tell us their numerical valuation of each wine!

Learning to Rank History

- Tradition: each field develops solutions ignoring all other fields
 - BIG topic in search engines
 - Joachims, T. (2002), "Optimizing Search Engines using Clickthrough Data", *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*
 - Tie-Yan Liu (2009), "Learning to Rank for Information Retrieval", *Foundations and Trends in Information Retrieval*, **3** (3): 225–331
- Huge survey: https://en.wikipedia.org/wiki/Learning_to_rank
 - Claims that logistic regression does not work
 - “Bill Cooper proposed **logistic regression** for the same purpose in 1992^[19] and used it with his **Berkeley** research group to train a successful ranking function for **TREC**. Manning et al.^[40] suggest that these early works achieved limited results in their time due to little available training data and poor machine learning techniques.”

Maybe Things Can Be Easy?

- If: most of the unobserved activation lists have fewer than two activations.
- Then:
 - Most of the unobserved activation lists are then identical to the observed selection lists.
 - Problem is solvable by direct item-wise logistic regression.
- Example: online shopping with low activation rates.

Maybe Things are Difficult?

- Presentation position may be a strong influence
 - Users may be biased to pick earlier presentation positions, forget earlier positions, or even not even try/look-at later positions!
 - Item quality may correlate with presentation position.
 - What other items are in the list or panel having an effect (such as high priced irrelevant alternatives).
- Data is noisy
 - User may make different choice when re-presented
- Data is low fidelity or censored
 - Selecting 1 out of 5 positions is only 2.3 bits of information
- No-select lists ***may be coming from an unknown mixture of two sources***
 - Motivated Shoppers (with intent to buy, so non-buying strong evidence against all presented wines).
 - Idle Browsers (with no intent to buy, so non-buying against any wine).
- These are not the same persona! Need per-user variables to sort these users out.
- Punishes highly desirable selections *more* than less desirable selections.

Our “Wine” Example

- Artificial Problem
 - Know answer
 - 3 explanatory variables: x1, x2, x3
- Visible Data
 - Panels of 5 wines tasted in one sitting
 - explanatory variables
 - selected (if any) wine
- Many repetitions over many customers thought to be in same persona cluster.

| | group | position | example_index | x1 | x2 | x3 | score | hidden_activation | selected |
|----|-------|----------|---------------|-------|------|------|-------|-------------------|----------|
| 0 | 0 | 0 | 7 | 0.00 | 1.00 | 1.00 | 0.295 | False | False |
| 1 | 0 | 1 | 2 | 0.00 | 0.00 | 0.10 | 0.257 | False | False |
| 2 | 0 | 2 | 3 | 0.00 | 1.00 | 1.00 | 0.304 | False | False |
| 3 | 0 | 3 | 8 | 0.00 | 1.00 | 1.00 | 0.288 | False | False |
| 4 | 0 | 4 | 10 | 0.00 | 1.00 | 1.00 | 0.348 | False | False |
| 5 | 1 | 0 | 1 | 1.00 | 6.00 | 0.00 | 0.546 | True | False |
| 6 | 1 | 1 | 0 | 12.00 | 1.00 | 0.00 | 0.796 | True | True |
| 7 | 1 | 2 | 3 | 0.00 | 1.00 | 1.00 | 0.326 | True | False |
| 8 | 1 | 3 | 7 | 0.00 | 1.00 | 1.00 | 0.309 | False | False |
| 9 | 1 | 4 | 2 | 0.00 | 0.00 | 0.10 | 0.210 | True | False |
| 10 | 2 | 0 | 6 | 0.00 | 1.00 | 1.00 | 0.349 | False | False |
| 11 | 2 | 1 | 0 | 12.00 | 1.00 | 0.00 | 0.817 | True | True |
| 12 | 2 | 2 | 5 | 0.00 | 1.00 | 1.00 | 0.297 | False | False |
| 13 | 2 | 3 | 10 | 0.00 | 1.00 | 1.00 | 0.327 | True | False |
| 14 | 2 | 4 | 8 | 0.00 | 1.00 | 1.00 | 0.327 | False | False |

The (Assumed) Data Generating Process

Evidence pattern on selection



User considers 5 alternatives



(unobserved) explanatory variables converted into activation probabilities (scores)



(unobserved) probabilities converted into “interested” activations check-marks



(observed) checked item with highest activation score is picked



0.349

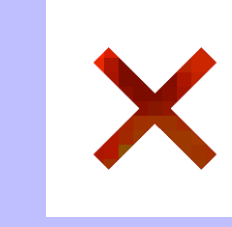
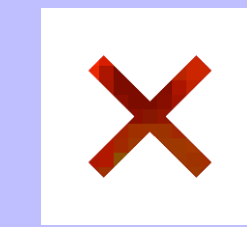
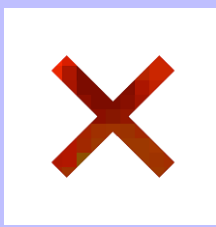
0.817

0.297

0.327

0.327

wish we knew this



activation list



selection list

experiment design

model structure

$f_{\beta}(x)$ (fit for β)

presentation censoring process

Solution

Stan Model

- Guess: β (coefficient vector including intercept), ζ (hidden per chosen selection vector noise term).
- Let $x_{i,j}$ denote the vector of explanatory values for the i th example's j th alternative

We model the hidden scores as being distributed normal($x_{i,j} \cdot \beta, \sigma$) (or $\zeta_i \sim \text{normal}(0, \sigma)$).

For lists with no selection model:

$$P[data_i | \beta, \zeta_i] \sim \prod_j (1 - \text{sigmoid}(x_{i,j} \cdot \beta))$$

(No selection arises from failing to select any of the alternatives.)

For lists with a selection s model:

$$P[data_i | \beta, \zeta_i] \sim \text{sigmoid}(x_{i,s} \cdot \beta + \zeta_i) \prod_{j, j \neq s} (1 - \text{sigmoid}(x_{i,j} \cdot \beta)(1 - \Phi(x_{i,j} \cdot \beta | x_{i,s} \cdot \beta + \zeta_i, \sigma)))$$

(For a selection to prevail it must be activated, and none of the alternative must simultaneously activate and exceed the selection's score. We (optionally) condition on an explicit ζ_i is to make the terms conditionally independent and justify multiplying.

- Set up Stan to sample β, ζ such that $P[\beta, \zeta]P[data | \beta, \zeta]$ is large. These are going to be such that $P[\beta | data]$ is large (i.e. this is the β implied by our observation data).
- Could add terms to account for rejections by uninterested browsers (i.e. whole lists rejected independent of item valuation).

Actual Stan Code

```
data {
  int<lower=1> n_vars; // number of variables per alternative
  int<lower=2> n_alternatives; // number of items per presentation list
  int<lower=1> m_examples; // number of examples
  array[m_examples] int<lower=0, upper=n_alternatives> pick_index; // which item is picked, 0 means no pick
  array[n_alternatives] matrix[m_examples, n_vars] x; // explanatory variables
}
parameters {
  real beta_0; // model parameters
  vector[n_vars] beta; // model parameters
  vector[m_examples] noise_in_pick_link;
}
transformed parameters {
  array[n_alternatives] vector[m_examples] link; // link values
  for (sel_j in 1:n_alternatives) {
    link[sel_j] = beta_0 + x[sel_j] * beta;
  }
}
model {
  // basic priors
  beta_0 ~ normal(0, 10);
  beta ~ normal(0, 10);
  noise_in_pick_link ~ normal(0, 0.1);
  // log probability of observed situation
  for (row_i in 1:m_examples) {
    if (pick_index[row_i] <= 0) {
      for (sel_j in 1:n_alternatives) {
        target += log1m(inv_logit(link[sel_j][row_i])); // non-activation odds
      }
    } else {
      for (sel_j in 1:n_alternatives) {
        if (sel_j == pick_index[row_i]) {
          target += log(inv_logit(link[pick_index[row_i]][row_i] + noise_in_pick_link[row_i])); // probability selection indicates
        } else {
          target += log1m(inv_logit(link[sel_j][row_i]) // probability potential spoiler indicates
            // probability potential spoiler outscores selection
            * (1 - normal_cdf(link[pick_index[row_i]][row_i] + noise_in_pick_link[row_i] | link[sel_j][row_i], 0.1)));
        }
      }
    }
  }
}
```

Calling Stan from R or Python

```
library(rstan)
library(jsonlite)

data <- fromJSON("rank_src_censored_picks.stan")

sample <- stan(
  file = "rank_src_censored_picks.stan", # Stan program
  data = data,                          # named list of data
  chains = 4,                           # number of Markov chains
  cores = 4,                             # number of cores (could use one per chain)
  refresh = 0,                           # no progress shown
  pars=c("lp__", "beta_0", "beta")      # parameters to bring back
)

draws <- as.data.frame(sample)
```

```
from cmdstanpy import CmdStanModel

# quiet down Stan
logger = logging.getLogger("cmdstanpy")
logger.addHandler(logging.NullHandler())

# instantiate the model object
model_comp = CmdStanModel(stan_file="rank_src_censored_picks.stan")
# sample high probability parameter settings
sample Stan = model_comp.sample(
    data="rank_src_censored_picks.stan",
    show_progress=True,
    show_console=False,
)

draws = sample Stan.draws_pd(vars=['lp__', 'beta_0', 'beta'])
```

What you Get From Stan

```
draws |>  
  head() |>  
  knitr::kable()
```

| lp__ | beta_0 | beta[1] | beta[2] | beta[3] |
|-----------|-----------|-----------|-----------|-----------|
| -1251.130 | -1.520818 | 0.2282623 | 0.2879370 | 0.3201965 |
| -1250.358 | -1.506511 | 0.2233696 | 0.2475490 | 0.1977965 |
| -1251.936 | -1.552996 | 0.1998336 | 0.2522806 | 0.3323120 |
| -1249.657 | -1.533951 | 0.2080136 | 0.2574112 | 0.3573067 |
| -1248.750 | -1.493253 | 0.2228508 | 0.2416577 | 0.3151417 |
| -1248.531 | -1.221375 | 0.2030706 | 0.2373745 | 0.0258106 |

```
nrow(draws)
```

```
## [1] 4000
```


Result:

Activation Probabilities

| example_index | | x1 | x2 | x3 | data generation process | model from visible selections | model from pair differences | Stan model MCMC | Stan model optimizer |
|---------------|---|------|-----|-----|-------------------------|-------------------------------|-----------------------------|-----------------|----------------------|
| 0 | 0 | 12.0 | 1.0 | 0.0 | 0.80 | 0.80 | 0.99 | 0.80 | 0.80 |
| 1 | 1 | 1.0 | 6.0 | 0.0 | 0.55 | 0.39 | 0.94 | 0.56 | 0.56 |
| 2 | 2 | 0.0 | 0.0 | 0.1 | 0.24 | 0.04 | 0.51 | 0.26 | 0.32 |
| 3 | 3 | 0.0 | 1.0 | 1.0 | 0.31 | 0.10 | 0.73 | 0.31 | 0.34 |

Notice the Stan models dominate.

Models Compared

- **Model from visible selections:** treat each item selection or non-selection as a positive or negative example. Also called “item-wise” ranking. A nice logistic regression solution for comparison.
- **Model from pair differences:** encode each pair in a list where one wine is picked and one is not as an example. A common “let’s be clever trick.”
- **Stan Model MCMC:** encode an entire probability model (including guessing unseen quantities) and then use a Markov chain to sample high likelihood values. The big dog.
- **Stan Model Optimizer:** use the above encoding as a “loss”, and hope something as simple as standard optimizer can find a high likelihood solution. “Cheap Stan” (could use other numeric platforms like PyTorch for this).

Details: Feature Encoding

$x(\text{item} = 6425, \text{position} = 2) =$

| 6425 | |
|----------------------|---------|
| fixed acidity | 6.0 |
| volatile acidity | 0.34 |
| citric acid | 0.29 |
| residual sugar | 6.1 |
| chlorides | 0.046 |
| free sulfur dioxide | 29.0 |
| total sulfur dioxide | 134.0 |
| density | 0.99462 |
| pH | 3.48 |
| sulphates | 0.57 |
| alcohol | 10.7 |
| is_red | False |
| posn_0 | 0 |
| posn_1 | 0 |
| posn_2 | 1 |
| posn_3 | 0 |
| posn_4 | 0 |

Feature table lookup

Encoding of presentation position. Could also encode demographics of participant Especially useful if we interact the demographic variables with the feature variables.

Details: Difference Encoding

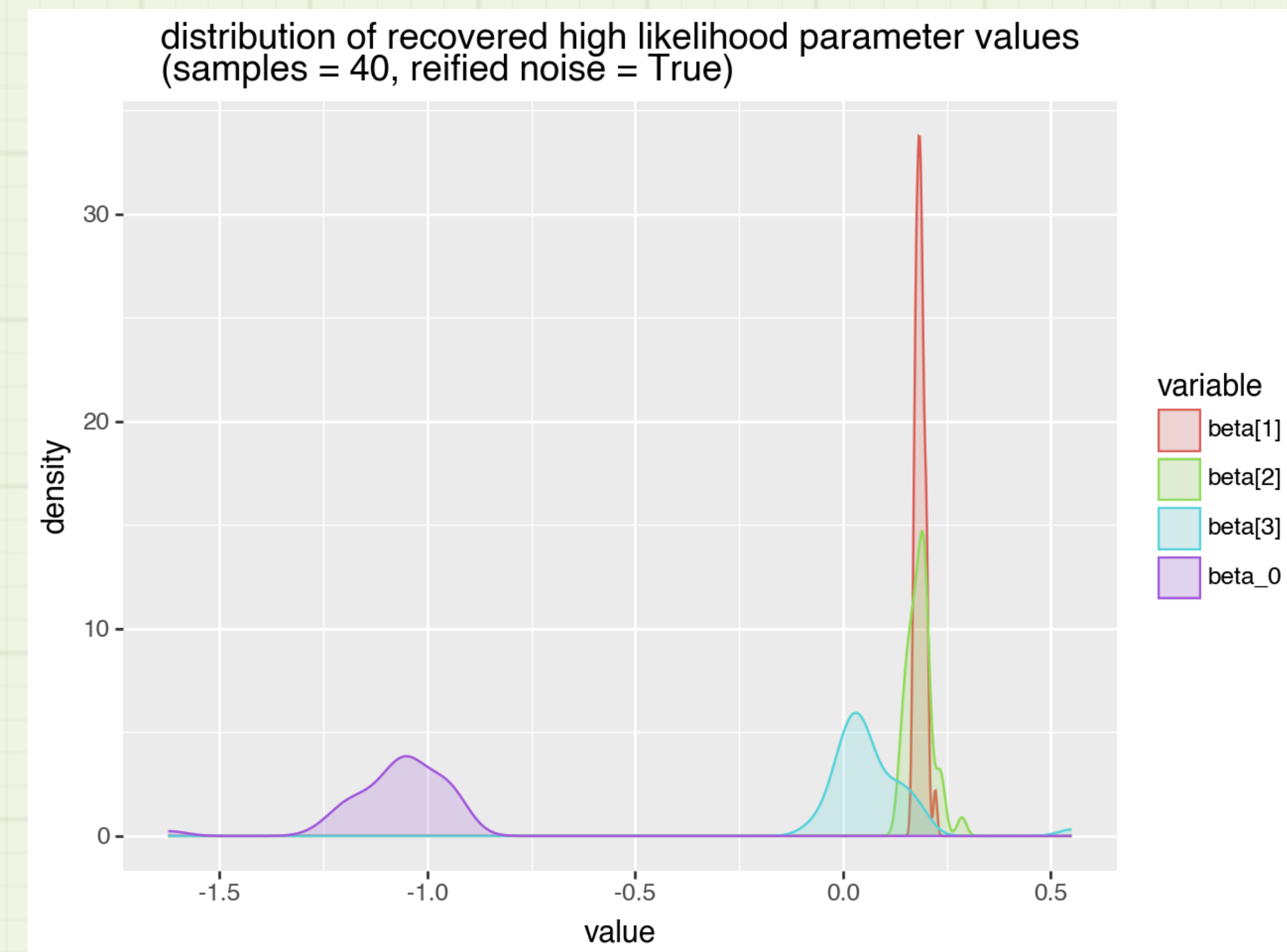
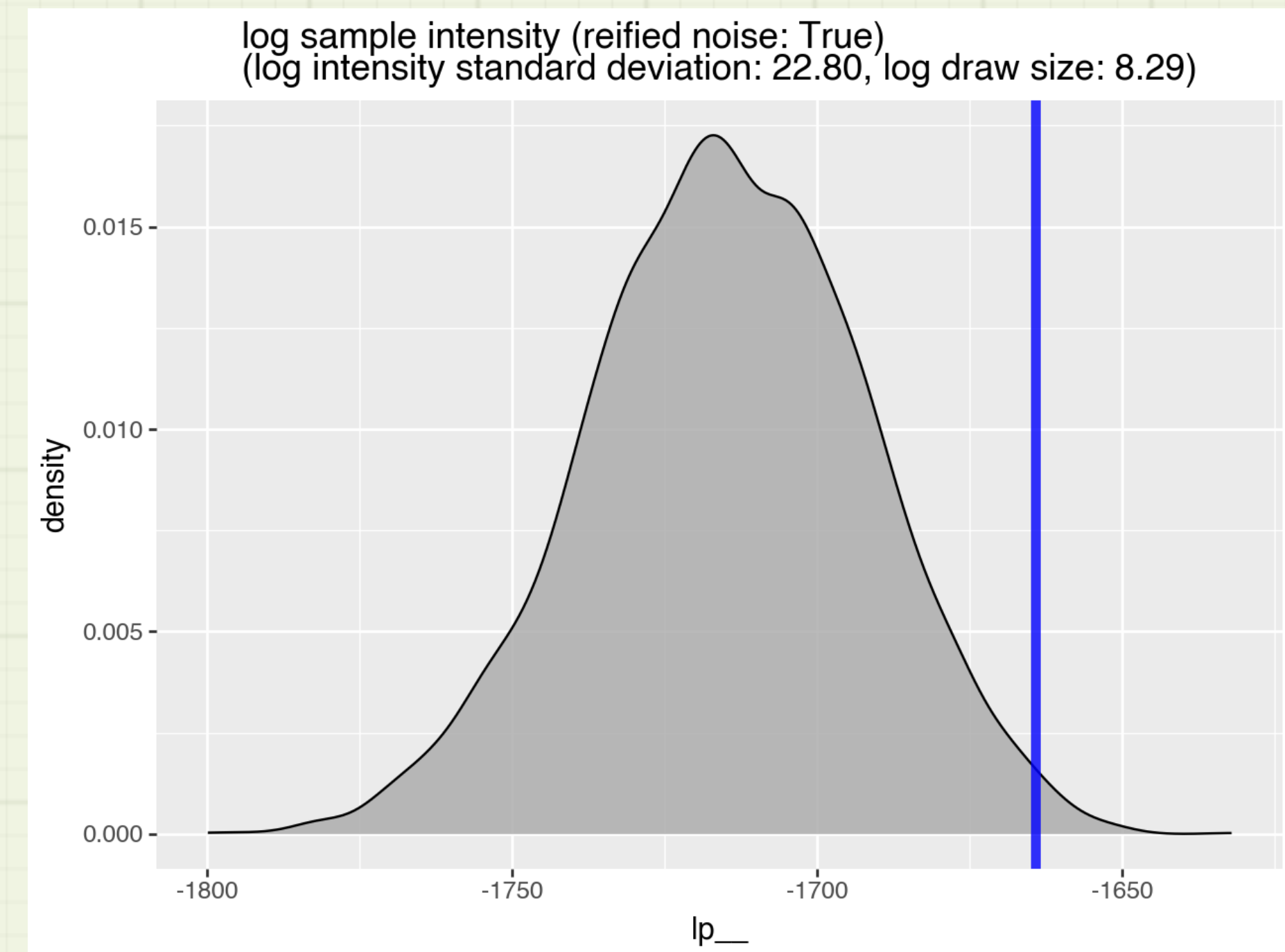
- Exploit linear structure to directly encode differences
 - Explanatory variables: encode comparison pairs (with different outcomes).
 - For example: item 6425 in position 2 **picked** and item 1569 in position 2 **not picked** encoded as:

$$x_i = x(\text{item} = 6425, \text{position} = 2) - x(\text{item} = 1569, \text{position} = 0)$$

- The subtraction enforces that features have the same interpretation in picks and non-picks.
 - Outcomes encodes as “True”.
 - Outcomes/dependent variables must vary
 - So also encode a “False” outcome for $-x_i$
- Encoding not natural or obvious.

Stan Diagnostics

- Check distribution of returned `__lp`.
 - Confirms Markov chain converged and model is identifiable.
 - The standard deviation should not be too much larger than the log of the number of samples.
 - This isn't always practical.
- Check distribution of parameter estimates with high `__lp`.
 - Confirms model is identifiable.
 - Confirms we are looking at a dominant estimate and not a mixture of incompatible estimates.



Our Results: Model Coefficients

| | model | beta_0 | x1 | x2 | x3 |
|---|-------------------------------|--------|------|------|-------|
| 0 | data generation process | -1.20 | 0.20 | 0.20 | 0.20 |
| 1 | model from visible selections | -3.13 | 0.34 | 0.39 | 0.50 |
| 2 | model from pair differences | 0.00 | 0.32 | 0.41 | 0.57 |
| 3 | Stan model MCMC | -1.05 | 0.19 | 0.18 | 0.05 |
| 4 | Stan model optimizer | -0.73 | 0.16 | 0.13 | -0.06 |

| example_index | | x1 | x2 | x3 | data generation process | model from visible selections | model from pair differences | Stan model MCMC | Stan model optimizer |
|---------------|---|------|-----|-----|-------------------------|-------------------------------|-----------------------------|-----------------|----------------------|
| 0 | 0 | 12.0 | 1.0 | 0.0 | 0.80 | 0.80 | 0.99 | 0.80 | 0.80 |
| 1 | 1 | 1.0 | 6.0 | 0.0 | 0.55 | 0.39 | 0.94 | 0.56 | 0.56 |
| 2 | 2 | 0.0 | 0.0 | 0.1 | 0.24 | 0.04 | 0.51 | 0.26 | 0.32 |
| 3 | 3 | 0.0 | 1.0 | 1.0 | 0.31 | 0.10 | 0.73 | 0.31 | 0.34 |

Not Just a Prevalence Issue

| | model | beta_0 | x1 | x2 | x3 | recovered prevalence |
|---|-------------------------------|--------|------|------|------|----------------------|
| 0 | model from visible selections | -1.78 | 0.34 | 0.39 | 0.50 | 0.38 |
| 1 | model from pair differences | -1.85 | 0.32 | 0.41 | 0.57 | 0.38 |

| example_index | | x1 | x2 | x3 | data generation process | model from visible selections | model from pair differences |
|---------------|---|------|-----|-----|-------------------------|-------------------------------|-----------------------------|
| 0 | 0 | 12.0 | 1.0 | 0.0 | 0.80 | 0.94 | 0.92 |
| 1 | 1 | 1.0 | 6.0 | 0.0 | 0.55 | 0.71 | 0.72 |
| 2 | 2 | 0.0 | 0.0 | 0.1 | 0.24 | 0.15 | 0.14 |
| 3 | 3 | 0.0 | 1.0 | 1.0 | 0.31 | 0.29 | 0.29 |

Conclusions

- There are examples where easy inference of preference doesn't work.
- Only Stan was powerful to let us pick our assumed presentation censoring process.
 - The other systems impose a presentation behavior.
- Stan is great for prototyping solution methods and experimenting with how much model structure and presentation hygiene you wish to capture.
 - Dropping the modeled errors made for a smaller model that was nearly as effective.
 - Can export inferred parameters and use them elsewhere.
 - Doesn't Stan in production.

Tools Used

- Stan, R, and Python
- All code and data shared here:

<https://github.com/WinVector/Examples/tree/main/rank>

rstan.md

rstan.Rmd

rank_src_censored_picks_reified_noise.stan

rank_src_censored_picks.stan

generate_example.ipynb

rank_data_censored_picks.json

LearningToRank.pdf

Thank you