Preference Inference Using Stan

John Mount, Ph.D.
Win Vector LLC
https://www.win-vector.com/

Bay Area Use R Group
https://www.meetup.com/r-users/events/303488652/
October 15, 2024

Slides, links, and code: https://github.com/WinVector/Examples/tree/main/rank

Win-Vector LLC

Slides, links, and code: https://github.com/WinVector/Examples/tree/main/rank

Who I am

- John Mount, General Partner at Win Vector LLC.
  - Co-author of *Practical Data Science with R*, 2nd edition, Manning, 2020.
  - Co-author of the `vtreat` R package for re-encoding high cardinality explanatory variables.
- Win Vector LLC is a statistics, machine learning, and data science consultancy and training organization.
  - Specialize in solution design, technology evaluation, and prototyping.
  - We help greatly speed up solution and production deployment.
  - Looking for some more engagements.
    - Please contact: jmount@win-vector.com .
- Please follow us on the Win Vector blog: https://win-vector.com/blog-2/

Win-Vector LLC

Outline

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

Win-Vector LLC

I am going to try to outline an interesting problem, show you the basics of using Stan, and the basics of taking benefits from Stan's solution.

The Problem

Example Problem

- 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 a numerical valuation of each wine!

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).

Formal Set Up

- 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 (unfortunately) only observe user ***behaviors***
  - Typical observation: we presented 5 alternatives and the user selected 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).
- Typically inferring hidden state from observations is very high value.
  - Stan can simulate inverting fairly complicated “hidden state to observations” processes.

Leaning 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 and used it with his Berkeley research group to train a successful ranking function for TREC. Manning et al. 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 two alternatives?
- Then:
  - This looks like the question structure from conjoint analysis
  - Most of the unobserved activation lists are then identical to the observed selection lists.
  - Problem is solvable by direct item-wise logistic regression.

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 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! Would need per-user variables to sort these users out.
  - Punishes highly desirable selections *more* than less desirable selections.

Win-Vector LLC

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 *the 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

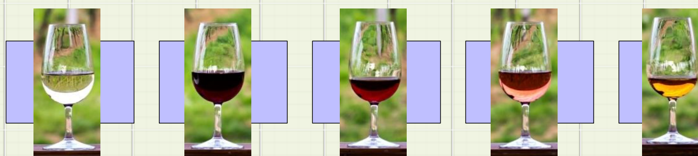
unobserved

The (Assumed) Data Generating Process

Evidence pattern on selection

experiment design

User considers 5 alternatives



model structure

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

0.349	**0.817**	0.297	0.327	0.327

wish we knew this

$f_\beta(x)$ (fit for β)

(unobserved) probabilities converted into "interested" activations check-marks

✗	✓	✗	✓	✗

activation list

presentation censoring process

(observed) checked item with highest activation score is picked

	✓			

selection list

Win-Vector LLC

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, *and unobserved intermediate state* conditioned on observed data.
    - Returns distributional answers.
- Fairly high dependency (requires C++ compiler and linker)
    - May eventually see competition from Torch based alternatives

Solution

(Approximate) Stan Model

- Define parameters: β, β_0 (coefficient vector and intercept) *and* ζ (hidden per chosen selection vector link value).
- Let $x_{i,j}$ denote the vector of explanatory values for the ith example's jth alternative
    - We model a hidden link scores as being distributed normal$(\beta_0 + x_{i,j} \cdot \beta, \sigma)$
    - We (**evilly**) approximate this many places as $\beta_0 + x_{i,j} \cdot \beta$
- For lists with no selection model, the probability of seeing the observed non-selection is:

$$\text{P}[data_i \,|\, \beta, \zeta_i] \sim \prod_j (1 - \text{sigmoid}(\beta_0 + x_{i,j} \cdot \beta))$$

- For lists with a selection s model, the probability of seeing the observed selection is:

$$\zeta_{i,s} \sim \text{normal}(\beta_0 + x_{i,j} \cdot \beta, \sigma)$$

$$\text{P}[data_i \,|\, \beta, \zeta_{i,s}] \sim \text{sigmoid}(\zeta_{i,s}) \prod_{j, j \neq s} (1 - \text{sigmoid}(\beta_0 + x_{i,j} \cdot \beta)(1 - \Phi(\beta_0 + x_{i,j} \cdot \beta \,|\, \zeta_i, \sigma))$$

- Set up Stan to sample β, ζ such that $\text{P}[\beta, \zeta]\text{P}[data \,|\, \beta, \zeta]$ is large.

No selection arrises from failing to select any of the alternatives.

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

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)));
        }
      }
    }
  }
}
```

incoming data

values to guess at

notational convenience

the criticism

Win-Vector LLC

Setting up the blocks is the trick. The main section is the model block, all other blocks are in service of the model block. We prefer vectorized operations and no control structures such as “for” or “if”. In our case we have them, but notice all control structures are conditional on data and not on inferred parameters. This means the scoring process is the same for all parameter values, and we are well within the design regime of Stan. Stan maximizes the hidden variable “target”.

Stan Encoding

- Stan code is to propose a desirability of the simultaneous settings of data and parameters.
  - This desirability is encoded in a variable called "target"
  - Stan samples parameter values proportional to exp(target).
- We add our criticisms to target:
  - Directly as in $target \mathrel{+}= \log(1 - \text{inv\_logit}(link))$
    - That is encoding: "our data has a non-select here, so we want the modeled probability of selection to be low."
  - Distributionally as in $beta \sim normal(0,10)$
    - Equivalent to $target \mathrel{+}= \text{normal\_lpdf}(beta \mid 0,10)$
    - Nicer looking.

Win-Vector LLC

Have to get comfortable with mixing probabilities and densities. Easiest to thin of the Stan code as a grouped criticism, and not worry over much what is prior and posterior conditioning.

Stan Code Again

```
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)));
        }
      }
    }
  }
}
```

the math

Win-Vector LLC

The math from 3 slides ago is only these few lines.

Calling Stan from R or Python

```r
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)
```

```python
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'])
```

Why That Works

- We want to maximize plausibility of parameters given data: P$[\beta, \zeta \,|\, data]$.
- By Bayes' Law P$[\beta, \zeta \,|\, data] =$ P$[\beta, \zeta]$P$[data \,|\, \beta, \zeta]$/P$[data]$.
  - Can ignore P$[data]$ as it is free of our parameter estimates.
  - So picking β, ζ such that P$[\beta, \zeta]$P$[data \,|\, \beta, \zeta]$ is large is the same as picking such that P$[\beta, \zeta \,|\, data]$ is large.
- Can invoke ideas such as the Bernstein-von Mises theorem to argue "large" is going to concentrate samples near the (unknown) true parameter value as we add more data.
  - (Technically better to argue that for a model with no data size dependent variables ζ.)
  - With enough data and flat enough P$[\beta, \zeta]$, we can even omit the P$[\beta, \zeta]$ term.
    - Structural mis-specification *more* risky than "wrong priors."

Win-Vector LLC

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.

Win-Vector LLC

In this table we have 4 types of examples, one in each row. The numbers to the right of the `x` columns are the probability the item is selected. So, good models should return numbers near the data generation process column. The models evaluated are listed in the next slide.

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).

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

Win-Vector LLC

This slide is what happens if we statistically adjust (or correct) the simple models to reproduce the correct prevalence. What happens is: not too much improvement, and the difference between the simple logistic model and difference model goes away.

Conclusions

- There are examples where easy inference of preference doesn’t work.
- 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.

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

Our Using Stan to Solve Problems Training Offering

- The series currently includes:
  - Dealing with range censored data, or tobit style regression.
  - Learning rank preferences from observed actions.
  - Time series with external explanatory variables.
- Contact jmount@win-vector.com for custom training and consulting.

Thank you

Slides, links, and code: https://github.com/WinVector/Examples/tree/main/rank