Prototyping Preference Inference Using Stan

John Mount
Win Vector LLC
https://www.win-vector.com/
August 29, 2024

Win-Vector LLC

Outline

- The problem
  - Some history
- Bayesian solution
  - Experiments/Results
- Simpler solutions
- Observations
- Conclusions/recommendations
- Next steps

The Situation

- Users (or user personas) have ***preferences*** (either implicit or explicit and unobserved/shared)
  - 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-personal data.
- We observe user ***behaviors***
  - Typical observation: we presented 5 alternatives and the user purchased one.
- 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 behaviors).

Example Problems

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

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

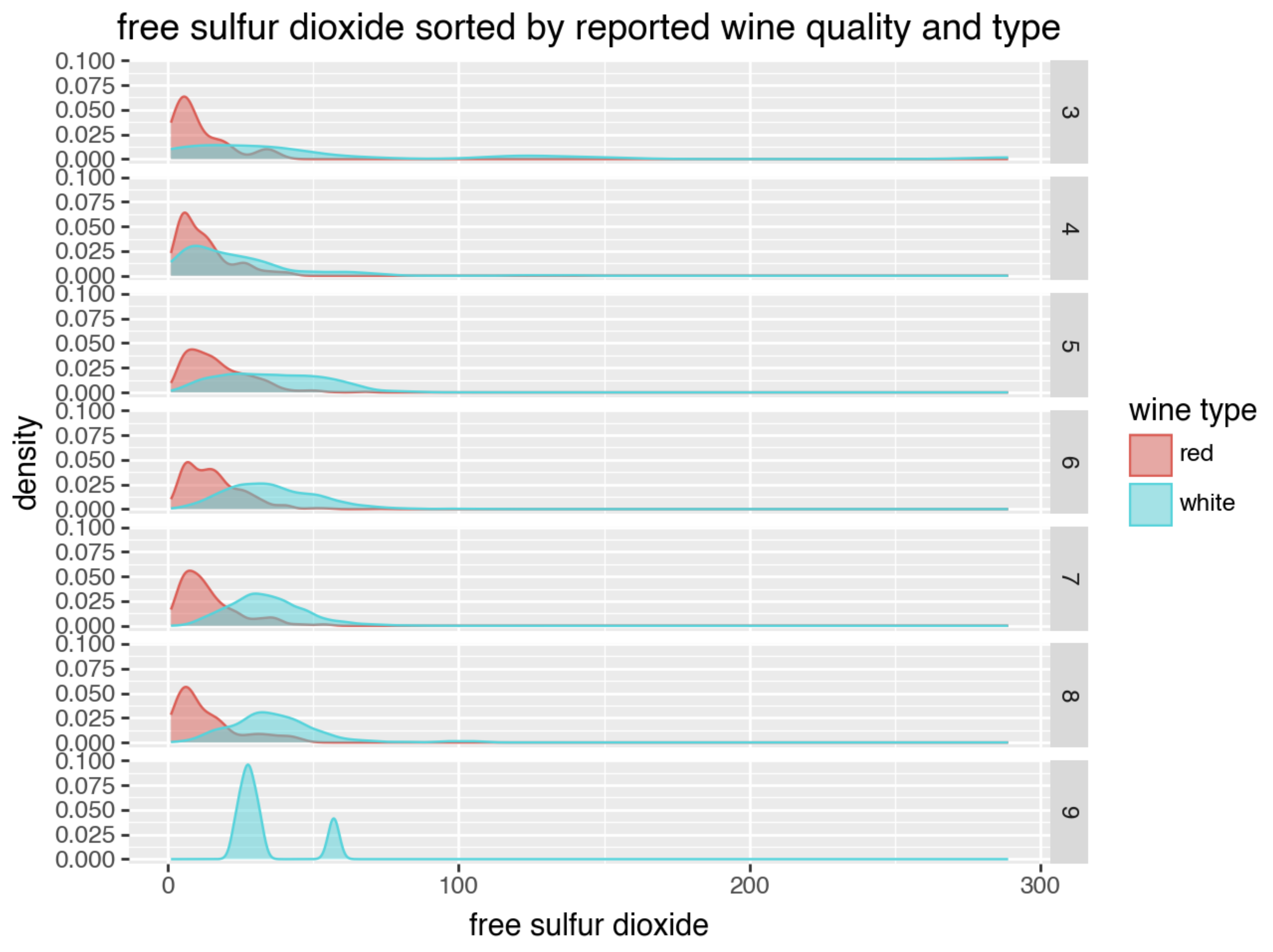
Win-Vector LLC

The Wine Example

Variables:

alcohol
chlorides
citric acid
density
fixed acidity
free sulfur dioxide *(shown as example)*
is_red *(we interact this with all other variables)*
pH
residual sugar
sulphates
total sulfur dioxide
volatile acidity

Not actually a good set of variables for the task (physical chemical instead of domain perceptual).

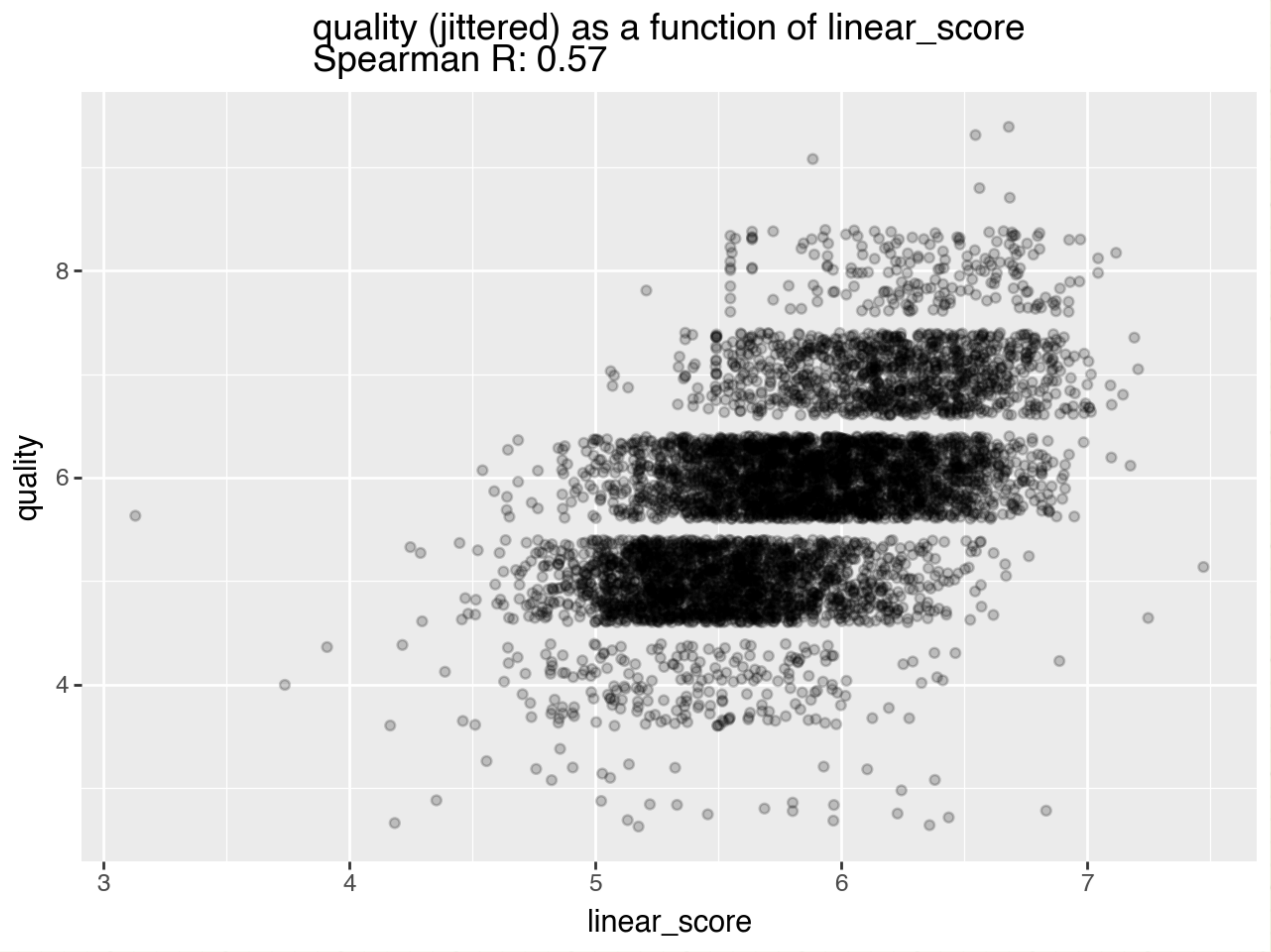


https://archive.ics.uci.edu/dataset/186/wine+quality

The Data we Wish For

User tells us their darn scores!

With 100 panels (lists or tastings) we observe heavily noisy and censored outcomes from about 500 wines out of our 6497.



(Above fit on all 6497 wines, without noise or list structure. Notice reported quality unfortunately isn't a linear function of our variables. Please remember 0.57 as "best possible" for this model structure.)

Win-Vector LLC

Confounding Issues

- Presentation position may be a strong influence
    - Users may be biased to pick in earlier presentation positions.
- Data is noisy
    - User may make different choice when re-presented
- Data is low information or censored
    - Selecting 1 out of 5 positions is only 2.3 bits of information
- Presentation is not an ideal experiment (influenced by business needs)
    - Ideal statistical procedure would be to present 2 alternatives and force a selection
        - Conjoint Analysis!
    - Item quality may correlate with position (would require an interaction to be introduced to the model).

My Intended Points

- The critical concern is having the right model structure.
  - Must approximate the data generation process.
  - Stan is great for prototyping inference strategies.
- The data presentation is statistical censoring
  - We can remove the censoring, but that doesn’t change if we have the right or wrong model.
  - The censoring increases the required amount of data
- There are several “right ways” to undo the censoring
  - Gives us some useful trade-offs

Leaning to Rank History

- 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
    - Econometrics has its own methods
    - Claims that logistic regression does not work
    - *Tons* of ink spilled on “what is the right way to undo the pick censorship?” (delaying working on the actual problem)
    - …
- Issues such as “point-wise” (each comparison is an event) versus “list-wise” (each presented list is an event) dominate problem design.
- Huge emphasis on efficiency of calculation.
    - Trade-off may be different a quarter of a century later.
- Huge survey: https://en.wikipedia.org/wiki/Learning_to_rank
    - “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.”

Win-Vector LLC

Our Goal

- Estimate the user (or user cohort) intrinsic preferences (independent of presentation).

- Eliminates some systems, such as **`xgb.XGBRanker`** (as it reproduces predictions over whole panels

- 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 is means data is in presentation and we `xgb.XGBRanker` is compute probability of selection with respect to the alternatives, not utility.)

Win-Vector LLC

A Bayesian Solution

- Suppose the user has a hidden valuation parameter β vector such that their valuation of a an item with features x_i is $f_\beta(x_i)$
  - Often this is realized as $f_\beta(x_i) = \beta \cdot x_i$.
- Further assume for a list of items $x_1, \ldots, x_5$ the user values item i with a value of $f_\beta(x_i) + e_i$ (e_i being a mean-zero noise term).
- Notice I have *not* chosen a probabilistic model!
  - Rank valuation is just an abstract number, utility, value, preference, or affinity.
- Let's introduce probabilities by modeling the user as picking the i such that this expression is maximized.
  - This formulation differs from the standard logistic solution in that we are not assuming a link function and error-rate, but instead a value denominated noise process.

Bayesian Reasoning

- We apply Bayes' Theorem.

Define:

$$Z_i = 1/\mathsf{P}[select(i) \mid x_1, \ldots, x_k]$$
$$select(i) = \wedge_{j,\, j \neq i} (\beta \cdot x_i + e_i > \beta \cdot x_j + e_j)$$

The trick in working with Stan:
make things conditionally independent.

Then:

$$\mathsf{P}[\beta \mid select(i), e_1, \ldots, e_k] = \mathsf{P}[\beta \mid e_1, \ldots, e_k]\ \mathsf{P}[select(i) \mid \beta, e_1, \ldots, e_k] \ / \ \mathsf{P}[select(i) \mid e_1, \ldots, e_k]$$ //Bayes' theorem

$$= Z_i\ \mathsf{P}[\beta]\ \mathsf{P}[\ \wedge_{j,\, j \neq i} (\beta \cdot x_i + e_i > \beta \cdot x_j + e_j) \mid \beta, \wedge_j\, e_j]$$ //expanding definitions, remove unused conditions

$$= Z_i\ \mathsf{P}[\beta] \prod_{j,\, j \neq i} \mathsf{P}[\beta \cdot x_i + e_i > \beta \cdot x_j + e_j \mid \beta, \wedge_j\, e_j]$$ //conditional independence

$$= Z_i\ \mathsf{P}[\beta] \prod_{j,\, j \neq i} \mathsf{P}[\beta \cdot x_i + e_i > \beta \cdot x_j + e_j \mid \beta, e_i, e_j]$$ //remove unused conditions

- (Z_i can be ignored in finding a maximum likelihood β, as it does not depend on β.)

- The point is: each of the checks become independent (can be written as a product) once we know e_i for the picked index i. If we explicitly draw e_i and β in our simulation, we can exploit the independence. We do not worry about the e_j, as they appear only once- so can't carry conditioning information between terms. We will call this formulation the list-wise model.

  - Or: I am going to type stuff into Stan, and try not to feel overly bad about it.

Win-Vector LLC

The Pick Data

Explanatory features are a combination of table lookup by item_id and presentation facts (i.e. presentation position).

	item_id_0	pick_value_0	item_id_1	pick_value_1	item_id_2	pick_value_2	item_id_3	pick_value_3	item_id_4	pick_value_4
0	1569	0	1754	0	6425	1	2780	0	2646	0
1	4390	1	2031	0	2692	0	4416	0	1913	0
2	599	1	1808	0	64	0	59	0	1671	0
3	1392	0	2324	0	5815	0	1819	1	4567	0
4	2063	0	6283	0	3610	1	2085	0	5610	0
5	2010	1	1465	0	6388	0	25	0	420	0
6	5903	1	1374	0	312	0	926	0	5467	0
7	5194	1	3651	0	1494	0	1749	0	5865	0
8	5946	1	4527	0	5988	0	3021	0	4821	0
9	6469	1	6044	0	2787	0	5786	0	3709	0

Feature Encoding

$$x(item = 6423, position = 2) =$$

	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.

Win-Vector LLC

Outcome Encoding: List-Wise Observations

Outcome (y):

Encodes as:

[0, 0, 0, 1, 0]

Simulates a single pick from a single tasting of each wine.

Point-Wise Observations

Outcome (y):

Simulates the (odd) situation of us guessing the user's preferred wine and then giving them 4 paired tastings of it against the 4 non-preferred wines. Notice wine 3 is tasted 4 times (all other wines tasted once).
Not what anyone wants, but easiest for most modeling systems.

Encodes as (independent events):

$$score_3 > score_0$$
$$score_3 > score_1$$
$$score_3 > score_2$$
$$score_3 > score_4$$

Win-Vector LLC

3 of Our Models

- Stan inspection patience model
    - Matches the data generation process
    - A bit complicated and brittle (so we won’t describe it here).
- Stan position utility mode
    - Models later positions as costing some utility or score.
    - Fairly flexible, even when it doesn’t match the data generation process.
- Logistic regression model
    - Less flexible (can’t imitate as many proposed generative processes)
    - Much faster and easier to deploy

Stan Position Utility Model

Pass one-hot encoded position indicators to the model as features. Models presentation position as a utility trade-off.

```
...
transformed parameters {
  …
  expect_picked = x_picked * beta;          // modeled expected score of picked item
  v_picked = expect_picked + error_picked;  // reified actual score of picked item
  expect_passed_1 = x_passed_1 * beta;      // modeled expected score of passed item
  expect_passed_2 = x_passed_2 * beta;      // modeled expected score of passed item
  expect_passed_3 = x_passed_3 * beta;      // modeled expected score of passed item
  expect_passed_4 = x_passed_4 * beta;      // modeled expected score of passed item
}
model {
    // basic priors
  beta ~ normal(0, 10);
  error_picked ~ normal(0, 10);
    // log probability of observed ordering as a function of parameters
    // terms are independent conditioned on knowing value of v_picked!
  target += normal_lcdf( v_picked | expect_passed_1, 10);
  target += normal_lcdf( v_picked | expect_passed_2, 10);
  target += normal_lcdf( v_picked | expect_passed_3, 10);
  target += normal_lcdf( v_picked | expect_passed_4, 10);
}
```

Win-Vector LLC

Logistic Model

- Trick 1: encode as a difference
  - Item 6425 in position 2 picked and item 1569 in position 2 not picked encoded as:

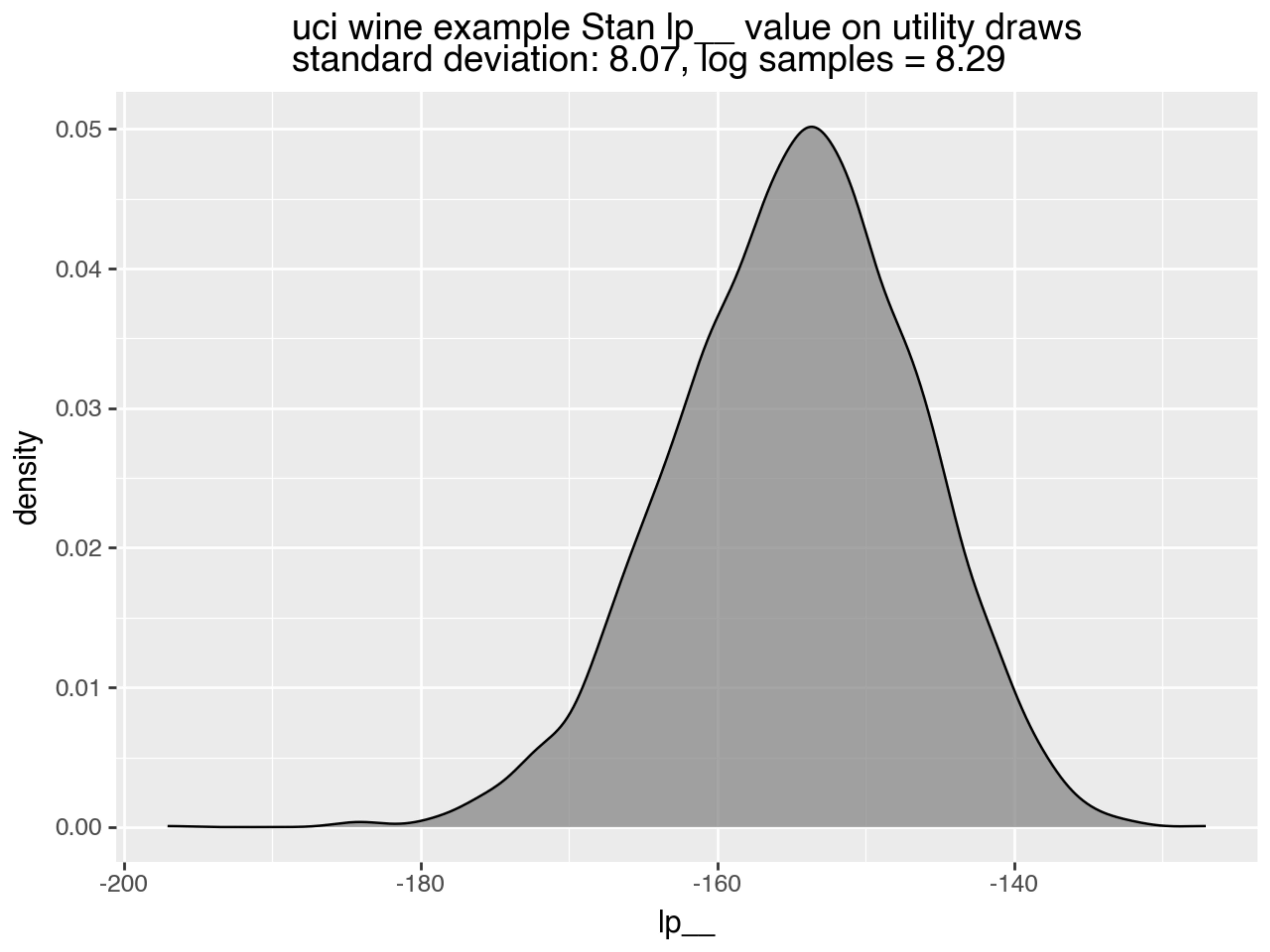
$$x_i = x(item = 6423, position = 2) - x(item = 1569, position = 0)$$
$$y_i = True$$

  - The subtraction enforces that features have the same interpretation in picks and non-picks. This is needed to have a model we can apply to items outside of panels and not knowing if they are picked or not. It also halves the number of model parameters, presumably making inference more statistically efficient (requiring less data). This is also why we are not using multinomial logistic regression, or multi class classification.
  - Problem: creates a data set with only “True” outcomes (can’t run fitter!).
- Trick 2: also encode reversal (in addition to seeing the winner winning, we saw losers lose)
  - Add in extra data rows of the form:

$$x_i = -\,(x(item = 6423, position = 2) - x(item = 1569, position = 0))$$
$$y_i = False$$

  - Now we have both True and False outcomes, and can try a logistic regression.
- Collect all the above rows as a logistic regression training set (one row for each pair with a different outcome per list).

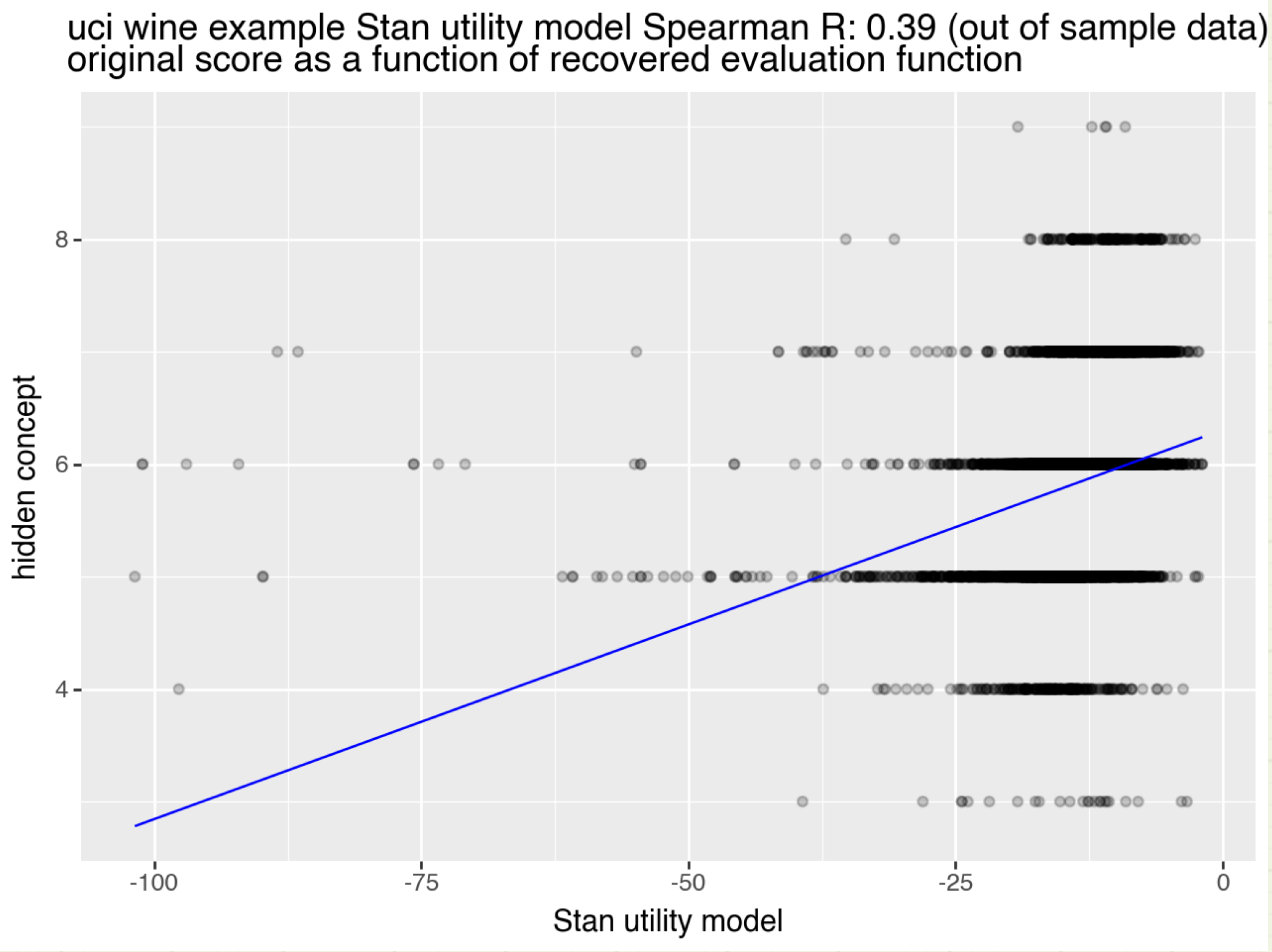
Stan Double Check



At least during development, you want to check lp__ is unimodal with a standard deviation not *too much* wider than +- log(n-samples) = log(4000) ~ 8.29. Though I have seen good runs that violate this.

Win-Vector LLC

Quality of Reproduced Preference Score



Recovered estimates of the *underlying* preference or utility scores. This may not look great, but remember we established 0.57 as best possible Spearman R for a linear model with this set of features. So Spearman 0.39 isn't *that* bad.

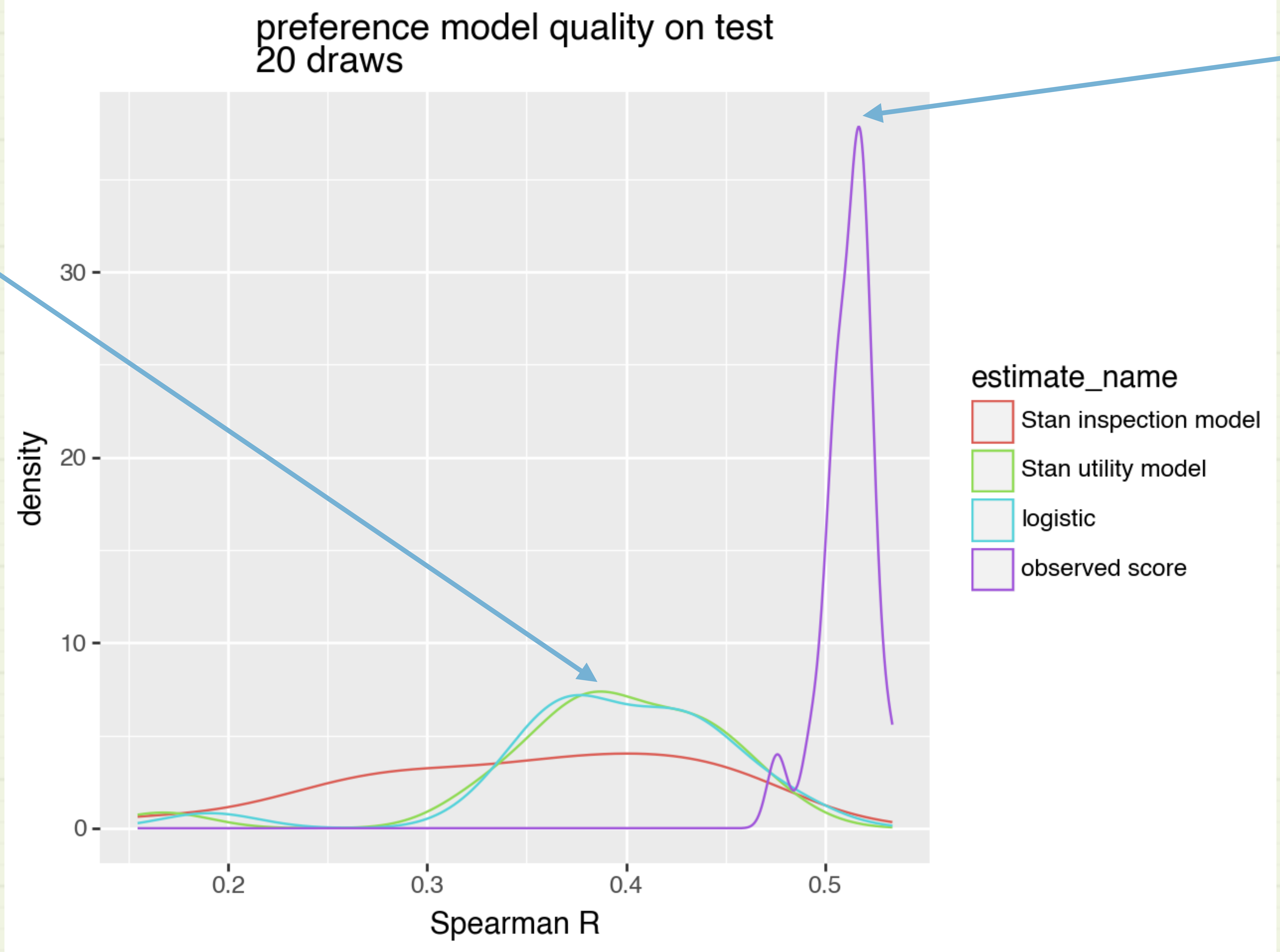
Our formulation works only on order data. So it is shift invariant (no reason to match absolute scores), and estimated scale (or differences) comes from the modeled signal to noise ratio.

Win-Vector LLC

Comparison of Methods

(100 panels, redrawn 20 times)

Various "try to infer from picks" methods.
What we can do in practice.
Has been implemented from just observed picks.



The "we wish users would tell us their scores" situation.
An upper bound on what is possible with the given model structure.
Can't be implemented from pick data.

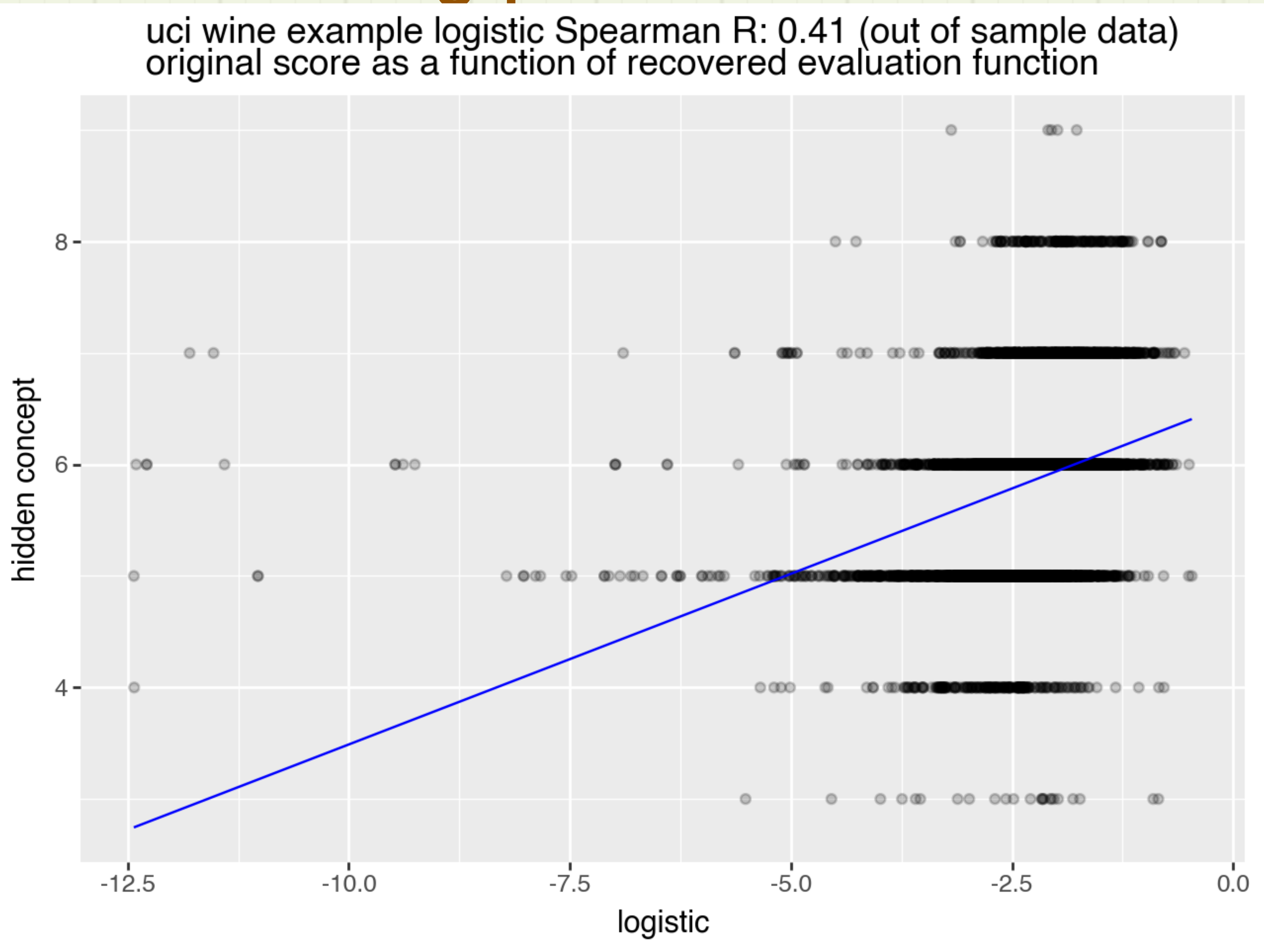
Gap goes away if we have more panels!

Win-Vector LLC

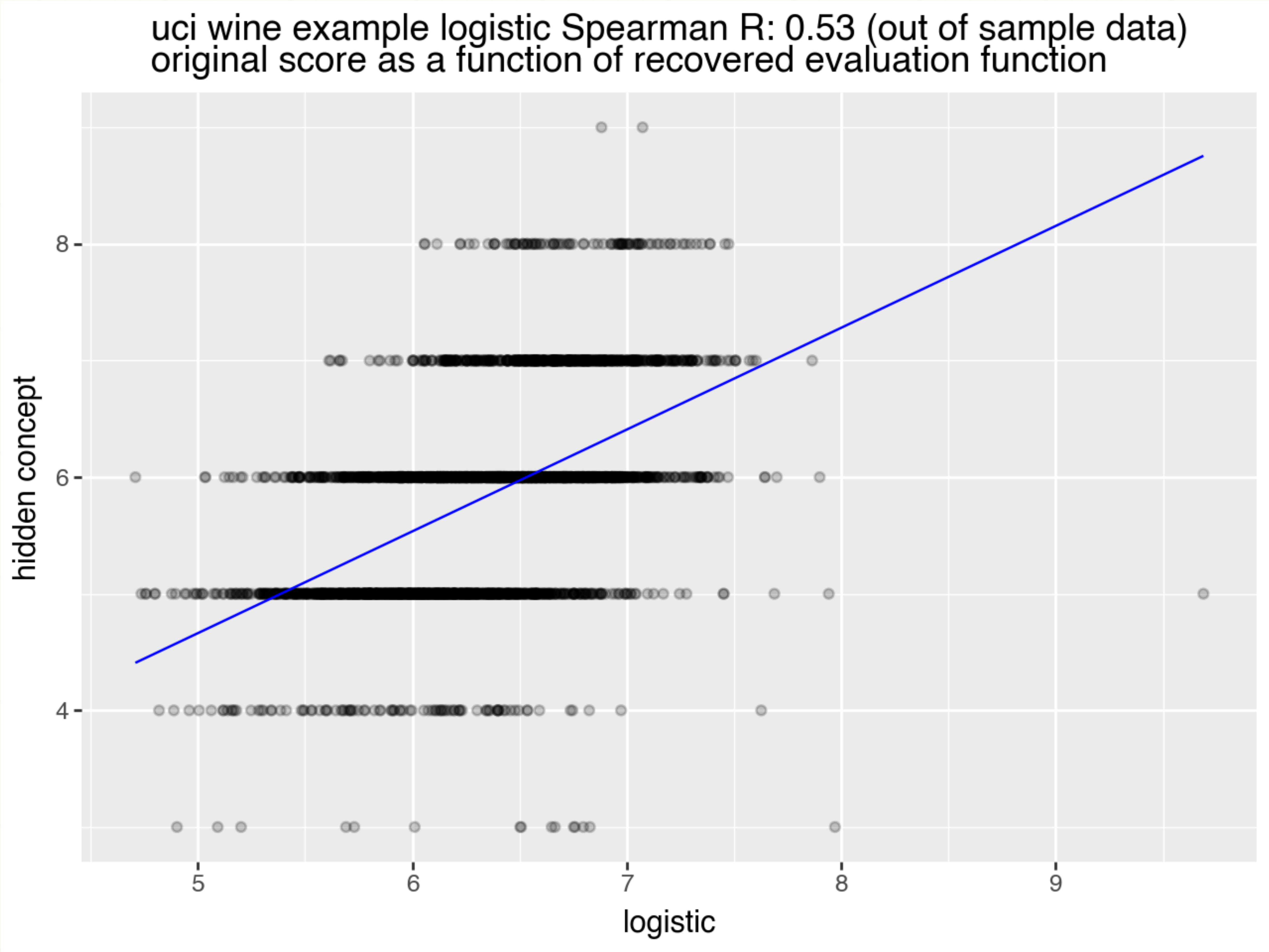
The Effect of More Data

Close to perfect (0.57)!

100 training panels



1000 training panels



Note: next slide shows we are not hiding behind a noisy evaluation.

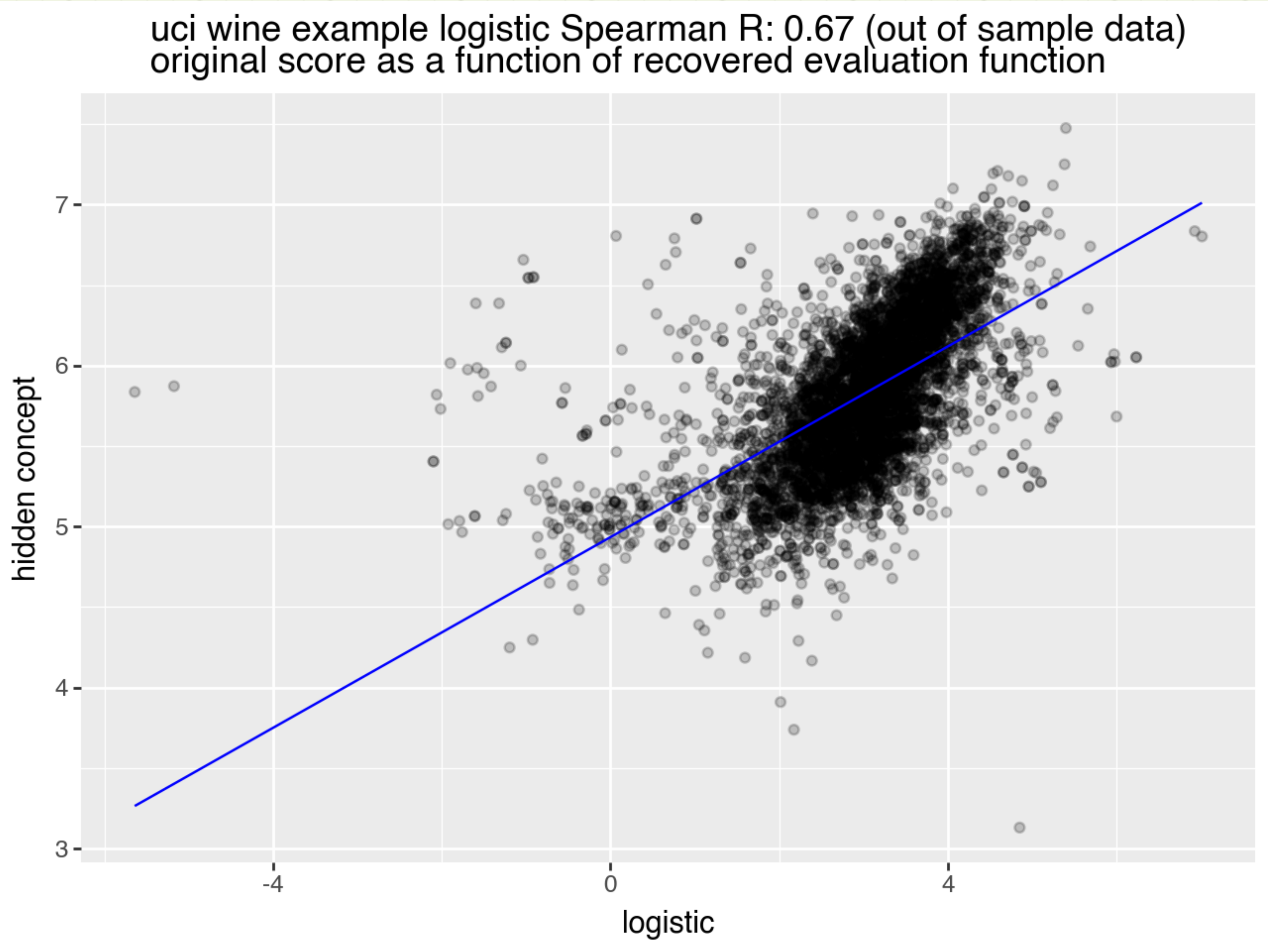
Win-Vector LLC

When We Have Correct Model Structure

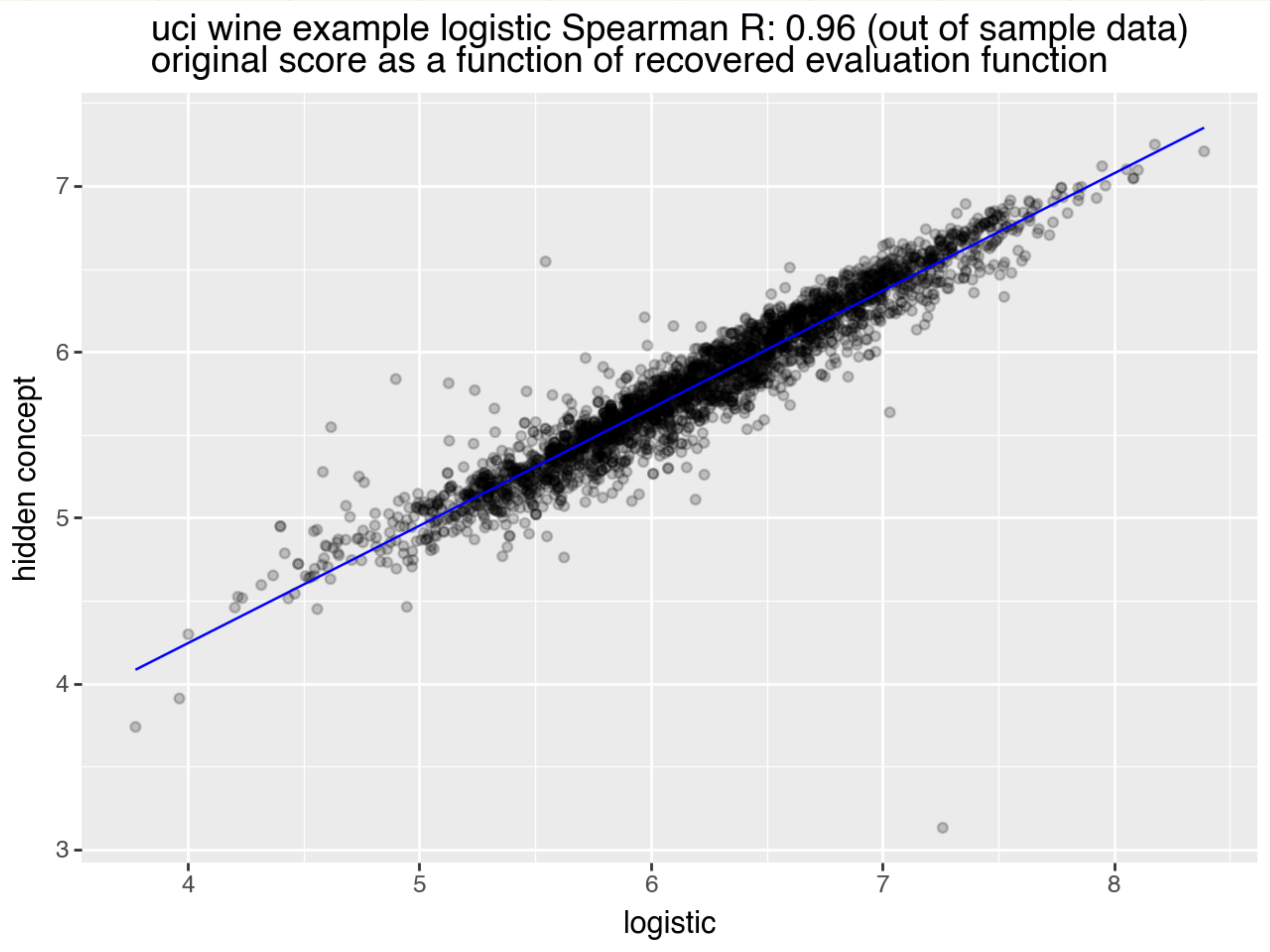
Replace training data with a linear model of user score (an easier problem).
Now (modulo noise) the concept is in our modeling space.

Nearly perfect (1.0)!

100 training panels



1000 training panels



Summary Results

	example_name	estimate_name	SpearmanR_all	SpearmanR_test	pick_auc	mean pick KL divergence	training lists	test lists	data_size	test_size
0	uci wine example	Stan inspection model	0.529049	0.528237	0.619100	inf	100	100	6497	6025
1	uci wine example	Stan utility model	0.682519	0.679902	0.729988	0.423077	100	100	6497	6025
2	uci wine example	logistic	0.671805	0.669006	0.724300	0.439139	100	100	6497	6025
3	uci wine example	observed score	0.947432	0.946485	0.625713	0.457901	100	100	6497	6025

	example_name	estimate_name	SpearmanR_all	SpearmanR_test	pick_auc	mean pick KL divergence	training lists	test lists	data_size	test_size
0	uci wine example	Stan inspection model	0.524222	0.522379	0.620048	0.460817	1000	1000	6497	3027
1	uci wine example	Stan utility model	0.536526	0.534661	0.763765	0.382364	1000	1000	6497	3027
2	uci wine example	logistic	0.535088	0.534574	0.761253	0.442014	1000	1000	6497	3027
3	uci wine example	observed score	0.567919	0.566747	0.612814	0.458842	1000	1000	6497	3027

Observations

- All 3 methods perform about as well as each other
    - With correct or incorrect model structure
- The path to a high quality fit is
    - Correct model structure
    - More data
    - And *apparently not* over-worrying on ranking technique.
        - Some non-linear structure can expressed in Stan.
        - We did not actually use the linear structure or coefficients of logistic regression anywhere (other than to get link-space predictions). Any other classifier that returns probabilities could be used with an logit transform of the returned predictions. However, the tree based classifiers are very sensitive to over fitting in this situation, even with larger data sets. In particular performance on the pick task further decouples from performance on unobserved score estimation.
    - Though some clever ideas can still be incorporated (such as training no examples past the winner).
- Previous criticism may have been mis-attributing poor fit due to bad model structure as poor fit due to “pick data” censoring.
    - Makes sense to try arbitrary classification models that return probabilities (with a logit transform after prediction).

Conclusions

- Problem structure likely more important than the issues of pick list presentation.
- Stan is great for prototyping solution methods and experimenting with how much model structure you wish to capture.
- Logistic regression may take some steps to justify, but works well and is fast.

Tools Used

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

    https://github.com/WinVector/Examples/tree/main/rank
- Example code can work from just selection data
  - Though it would have no base score to compare to.
  - Can compare to proxy score: ability to reproduce picks.

Next Steps

- See if the 3 solutions (Stan list-wise, Stan point-wise, logistic regression) behave similarly on data from chosen domains (where we may not know ground truth).
    - Code works from picks alone
        - hidden preferences used only to generate picks and check inference quality
        - can do different length panels by encoding a “never picked” extra indicator variable
- Try other classification models! (no reason to limit to linear structure)
- Implement “signal attenuation due to pick data” estimator tool.

Thank you