Tie Handling Is Part of the Evaluation Protocol: An Order-Invariance Audit for Tie-Heavy Recommender Scores

Chengkun Guo[1], Han Chen[2,*], Yilin Zhu[3] and Yingrui Li[2]

[1]Independent Researcher, Bethlehem, Pennsylvania, USA
[2]Independent Researcher, Washington, DC, USA
[3]Independent Researcher, Seattle, Washington, USA

Abstract

Offline top-k evaluation often ranks one held-out relevant item together with sampled negatives. When several candidates receive exactly the same score, the tie-breaking rule becomes part of the ranking. A common implementation stores the relevant item first and then applies a stable sort, which preserves input order among equal scores; the relevant item therefore wins every tie. We call an evaluator *row-order invariant* when permuting the input candidates without changing their identities, labels, or scores leaves the final ranking unchanged. We audit this property by holding candidates and scores fixed and changing only the tie-breaking rule. On 30,000 Amazon Beauty & Personal Care rows, NDCG@10 for a rating-weighted attribute-overlap score is 0.85 under input-order tie-breaking. A deterministic hash tie-break based on user and item IDs lowers it to 0.17. The exact expectation under uniform random tie-breaking closely matches the mean over 100 independent hash seeds, while a residualized attribute score with few exact ties is nearly unchanged. MovieLens Tag Genome shows the same pattern for an attribute-overlap score, whereas item popularity is nearly unchanged. We derive expected Hit Rate and NDCG at cutoff k when the relevant item is randomly ordered among candidates with the same score, and we provide a practical reporting checklist. The same issue can occur in sampled or full-catalog evaluation whenever exact ties affect top-k membership or rank.

Keywords

recommender systems, offline evaluation, sampled metrics, score ties, reproducibility, evaluation protocols

CEUR-WS.org/Vol-4341/paper2.pdf

1. Introduction

Offline recommender results depend on the complete evaluation protocol, not only on the metric formula. Relevant choices include the data split, candidate set, negative-sampling procedure, relevance definition, ranking rule, cutoff, and aggregation method [1, 2, 3, 4, 5]. One choice is often left implicit: how are candidates ordered when their scores are exactly equal?

Consider an evaluation row that stores the held-out relevant item first, followed by sampled negatives. If all candidates receive score zero and the evaluator applies a stable descending sort, the list does not move: the relevant item is ranked first even though the score did not distinguish it from any negative. Reordering the same candidates before sorting can produce a different metric without changing the model output.

We study this dependency through *row-order invariance*. An evaluator is row-order invariant if every permutation of the input candidates that preserves item identities, relevance labels, and scores produces the same final ranking. To test the property, we hold each candidate set and its scores fixed and change only the rule used to order exact ties. Thus, any resulting metric difference is caused by tie handling rather than training, candidate sampling, or scoring.

Prior work has already shown that tie-breaking can affect information-retrieval and recommender-system evaluation [6, 7, 8, 9]. Our focus is a specific systematic interaction in recommender evaluation:

FRAME'26: Methodology First - Rethinking Research Assessment in RecSys Workshop, September 28, 2026, Minneapolis, Minnesota, USA

*Corresponding author.
guochk@gmail.com (C. Guo); hc414t@gmail.com (H. Chen); yilin.zhu2@gmail.com (Y. Zhu); el516y@gmail.com (Y. Li)

Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-26

a relevant item placed at a fixed position, a tie-heavy score, and a stable sort that reuses input order as an undeclared secondary ranking criterion.

This paper makes three contributions. First, it formalizes row-order invariance and derives the expected Hit Rate and NDCG when the relevant item is uniformly placed within an exact-score tie. Second, it compares input-order and deterministic hash tie-breaking in both domains; for Amazon, it also reports the exact expected metric under uniform random tie-breaking and results from 100 independent hash seeds. Third, it gives a six-part reporting checklist covering the candidate set, candidate order before sorting, tie prevalence, tie-breaking, permutation tests, and sensitivity analysis.

2. Related Evaluation Methodology

Sampled and full-catalog evaluation define different candidate sets and can produce different metric values or system orderings [10, 4]. Our analysis begins after that candidate set has been declared. It asks whether the evaluator gives the same result when the same candidates and scores are presented in a different order.

Tied-score evaluation has a substantial history in information retrieval. McSherry and Najork derive efficient metric computation in the presence of ties [6]. Cabanac et al. show that an uncontrolled document-identifier tie-break can change TREC scores [7], and Lin and Yang study the effect of ties on repeatability across ranking implementations [9]. Tie-breaking has also been examined in recommender comparisons [8]. We build on this literature by isolating the interaction among relevant-item-first rows, stable sorting, and exact ties while holding candidate rows and scores fixed.

Candidate order also matters in listwise large-language-model ranking. Recent work studies order sensitivity and position-invariant listwise reranking [11, 12]. In those settings, changing candidate presentation can change the model's output scores or ranking. Our setting is downstream: scores are already fixed, and only the evaluator's treatment of equal scores changes.

Position bias in logged feedback is a different problem [13, 14, 15]. It acts during data collection and learning and can amplify item popularity through feedback loops, whereas the audit here acts after candidate scores have been produced.

3. When Tie-Breaking Changes the Metric

3.1. Candidate rows and exact-score ties

For a user u, let $C_u = (i_1, \ldots, i_m)$ be the ordered candidate list supplied to the evaluator, let $s_u(i)$ be the score of candidate i, and let $y_u(i) \in \{0, 1\}$ indicate relevance. The experiments use one relevant item and $m = 31$ candidates per row, but the definitions are general.

After sorting by descending score, a *tie block* is a maximal consecutive group of candidates with the same score. A tie block *crosses the top-k boundary* when some of its items are above rank k and others are below it. In that case, the tie-breaking rule can determine which items appear in the top-k ranked list. Even when the entire tie block is within the top k, the tie-breaking rule can still change position-sensitive metrics such as NDCG.

A *stable sort by score* alone is not row-order invariant because equal-score candidates retain their input positions. A deterministic *hash tie-break* is row-order invariant when it sorts equal-score candidates by a reproducible key computed from a disclosed seed, user ID, and item ID, using item ID as the final fallback if two candidates receive the same key. The key must not depend on the candidate's column or storage position.

3.2. Expected metrics under uniform random tie-breaking

Suppose a candidates have scores strictly larger than the relevant item, and the relevant item belongs to a tie block of size t. When the relevant item is first in the input row, a *stable sort* gives it rank $a + 1$.

Under *uniform random ordering* within the tie block, its rank is uniform on $\{a+1, \ldots, a+t\}$. With one relevant item per row,

$$\mathbb{E}[\text{Hit@}k] = \frac{\max\{0, \min(t, k-a)\}}{t}, \tag{1}$$

$$\mathbb{E}[\text{NDCG@}k] = \frac{1}{t} \sum_{r=a+1}^{\min(a+t,k)} \frac{1}{\log_2(r+1)}. \tag{2}$$

With one relevant item (the positive) and 30 negatives all tied ($a = 0, t = 31$), placing the relevant item first gives Hit@10 and NDCG@10 equal to 1. *Uniform random tie-breaking* gives expected Hit@10 $= 10/31 = 0.3226$ and expected NDCG@10 $= 0.1466$. The difference follows mechanically from the tie block; it is not evidence that the score identified the relevant item.

Uniformly permuting the complete candidate row independently of relevance before applying a stable sort is equivalent, in expectation, to uniform random ordering within each tie block. The metric from a single permutation is therefore an unbiased estimate of the expectation above, but it remains noisy and can change from run to run. Repeated random tie-breaking or direct analytic averaging gives a clearer summary.

3.3. Practical tie-breaking choices

We compare four ways to handle exact ties:

1. **Input-order tie-breaking.** Sort by score and keep the original candidate order when scores are equal. The result is deterministic but can depend on row construction.
2. **Deterministic hash tie-breaking.** Within each tie, sort by a reproducible key computed from a disclosed seed, user ID, and item ID, using item ID as the final fallback. This gives one repeatable ranking independent of input order.
3. **Repeated random tie-breaking.** Break ties with different random seeds and report the metric's mean and variation across runs. This estimates the result under random tie-breaking.
4. **Exact expectation under uniform random tie-breaking.** Compute the average metric over all possible positions of the relevant item within its tie block. In the one-relevant-item setting, Equations 1 and 2 give this value exactly.

The first method uses input order; the other three do not. Deterministic hashing selects one repeatable ranking. Repeated random tie-breaking estimates the average over random tie orders, while the analytic method computes that average exactly in the one-relevant-item setting.

4. Audit Design

4.1. Candidate sets and scores

Table 1 summarizes the two audits. Every candidate row stores the held-out relevant item first and then 30 sampled negatives before sorting. The candidate-sampling seed is 42 in both domains.

Amazon Beauty & Personal Care. We use Amazon Reviews 2023 [16], split chronologically within each user. Negatives exclude items in the user's training and validation history, but the sampler does not enforce uniqueness among the 30 negatives in a row. Three binary lexical proxies extracted from product metadata are used: *fragrance-free*, *cruelty-free*, and *sulfate-free*. Let H_u denote user u's training history, and let $A_i \in \{0,1\}^3$ be item i's attribute vector. For interaction $r \in H_u$, let i_r be its item and let $w_r = \text{clip}((\text{rating}_r - 1)/4, 0, 1)$ be its rating weight. The score named `raw_count` in the artifact is

$$s_{\text{overlap}}(u, i) = \left(\sum_{r \in H_u} w_r A_{i_r} \right)^{\top} A_i.$$

Table 1
Audit setup. The Amazon audit uses a deterministic sample of 30,000 rows from 2,073,945 held-out interactions; MovieLens uses 10,000 test users. The last score listed for each dataset is a diagnostic with little sensitivity to tie-breaking; the Amazon and MovieLens diagnostic scores are constructed differently.

Dataset	Candidate set	Audited scores
Amazon Beauty & Personal Care	One relevant item and 30 sampled negatives; 389,478 catalog items	Rating-weighted attribute overlap; centered attribute overlap; residualized attribute score
MovieLens 25M + Tag Genome	One relevant item and 30 unique sampled negatives	Unweighted tag-attribute overlap; centered tag overlap; item popularity

It measures the rating-weighted overlap between the candidate's attributes and attributes appearing in the user's history. Interactions with higher ratings receive larger weights. Simply counting the number of historical items would be constant across all candidates for a user and therefore could not rank them. The centered variant uses $A_i - \bar{A}$ for both historical items and candidates, where $\bar{A}$ is the catalog-wide mean attribute vector.

We include the Amazon residualized score because it has few exact ties and therefore provides a stability check when the primary score already determines the order. The score removes two predictable components before computing the same type of user–candidate dot product. For items, let X_i be the product-metadata text after removing predefined phrases that directly indicate the target attribute. The residual $R_i = A_i - \hat{p}(X_i)$ subtracts the attribute probability predicted from X_i; residuals are set to zero where $\hat{p}(X_i) > 0.95$. For interactions, three-fold cross-fitting predicts each training weight w_r from a smoothed user baseline and user-specific effects for the item's store, category, and price groups, without using the row being predicted. Let $\hat{c}_r$ denote this out-of-fold prediction for interaction r. The user profile is

$$\sum_{r \in H_u} \big(w_r - \hat{c}_r\big) R_{i_r},$$

and the candidate score is its dot product with R_i.

MovieLens 25M. Ratings of at least four stars are treated as positive [17]. Six Tag Genome concepts are thresholded at 0.7 to form binary item attributes: action, classic, visually appealing, based on a book, violence, and sci-fi [18]. The attribute-overlap score counts these attributes over qualifying training interactions and takes the dot product with the candidate's attribute vector. We also evaluate its centered version and item popularity. We include item popularity because its relevant-item ranks are nearly unchanged across the two tie-breaking rules.

4.2. Tie-breaking implementations and metrics

The deterministic rule hashes a fixed seed, user ID, and item ID into an unsigned 64-bit integer, then uses item ID as a final fallback. The Amazon seed is 20260316 and the MovieLens seed is 20260318. The exact mixer and tests are included in the artifact.

The original Amazon evaluator converted the 64-bit hash to 32-bit floating point before sorting. The reference implementation keeps the integer hash. On the reconstructed Amazon rows, the two implementations produce identical aggregate metrics. One row contains a collision between distinct item IDs after float32 conversion; it changes the internal order of two tied candidates but not top-10 membership or the relevant item's rank. The integer implementation remains unchanged when candidate order is permuted.

We report NDCG@10 and Hit Rate@10, the fraction of rows whose relevant item appears in the top 10. With one relevant item per row, Hit@10 equals Recall@10. We also measure the fraction of rows containing any exact tie, the fraction with a tie crossing the top-10 boundary, and the fraction in which the relevant item belongs to a tie.

4.3. Reproducibility

For Amazon, preserved historical inputs allow deterministic reconstruction of the 30,000 audited rows and their candidate-level scores. The reconstructed arrays reproduce all 84 aggregate values from the original run exactly, with maximum absolute difference zero. We use those arrays for the integer-hash, analytic-expectation, 100-seed, collision, and permutation analyses.

For MovieLens, we report the aggregate values unchanged from the accepted version. The paired aggregate result file was retained, but the row-level candidate and score arrays were not, so the additional row-level analyses are reported only for Amazon.

The accompanying artifact provides the evaluator, reconstructed Amazon rows and scores, tests, configurations, and provenance records: github.com/Guoo123/tie-evaluation-audit.

5. Results

5.1. Cross-domain sensitivity to input order

Table 2 reports the main controlled comparison. Candidate identities, labels, sampled negatives, primary scores, and metric code are fixed within each row; only exact-score tie-breaking changes.

Table 2
NDCG@10 under input-order and deterministic hash tie-breaking. For Amazon, the integer implementation yields the same aggregate values as the original float32 implementation.

Dataset	Score	Input order	Hash tie-break	Change
Amazon Beauty	Weighted attribute overlap	0.8474	0.1702	-0.6772
Amazon Beauty	Centered attribute overlap	0.7103	0.1627	-0.5476
Amazon Beauty	Residualized attribute score	0.1689	0.1685	-0.0004
MovieLens	Tag-attribute overlap	0.8380	0.2332	-0.6048
MovieLens	Centered tag overlap	0.8406	0.2341	-0.6065
MovieLens	Item popularity	0.5905	0.5905	-2.2×10^{-5}

For the Amazon weighted attribute-overlap score, every audited row contains an exact tie, 99.94% contain a tie crossing the top-10 boundary, and the relevant item belongs to a tie in 98.84% of rows. NDCG@10 falls from 0.8474 to 0.1702, while Hit@10 falls from 0.9997 to 0.3526. The centered score is similarly sensitive. The residualized attribute score has an exact tie in only 0.39% of rows and changes by 0.0004 in NDCG@10.

MovieLens shows the same aggregate pattern in a different item domain. NDCG@10 for tag-attribute overlap changes from 0.8380 to 0.2332, and Hit@10 changes from 0.9390 to 0.4358. The centered score behaves similarly. Item popularity is unchanged to four decimals, showing that changing tie-breaking has little effect when ties do not change the relevant rank.

5.2. Deterministic, randomized, and expected tie-breaking

Table 3 compares the three input-position-independent treatments alongside the input-order reference on the reconstructed Amazon rows. The exact expected metric under uniform random tie-breaking and the mean over 100 independent hash seeds agree closely. The deterministic hash selects one reproducible ordering and is therefore not required to equal that expectation, although it is close in these data.

The comparison separates two practical goals. Deterministic hash tie-breaking gives a repeatable ranking suitable for a benchmark implementation. The expected metric over ties reports the average over all unresolved positions inside a tie block. Repeated hashes provide a direct numerical check of the analytic result and reveal seed-to-seed variation.

Table 3
Amazon NDCG@10 under four tie treatments. "Expected over ties" analytically averages over all positions of the relevant item within each tie block under uniform random tie-breaking. "100 hashes" reports mean ± standard deviation over independent hash seeds.

Score	Input order	Hash tie-break	Expected over ties	100 hashes
Weighted attribute overlap	0.8474	0.1702	0.1693	0.1694 ± 0.0012
Centered attribute overlap	0.7103	0.1627	0.1614	0.1615 ± 0.0011
Residualized attribute score	0.1689	0.1685	0.1685	0.16851 ± 0.00003

6. Reporting Tie-Heavy Evaluation

Tie-breaking can change which items enter the top-k list and where relevant items rank within it. Ties across the cutoff can affect Hit@k, while ties entirely within the top k can still affect NDCG@k. The following checklist describes what to report about candidate construction and tie handling.

1. **Candidate set.** State whether evaluation is full-catalog or sampled, the number of candidates, the negative-sampling distribution, exclusions, replacement or deduplication behavior, and seeds.
2. **Candidate order before sorting.** State whether the relevant item occupies a fixed position and whether rows are independently shuffled.
3. **Tie prevalence.** Report the fraction of rows with any exact tie, with the relevant item in a tie, and with a tie crossing the top-k boundary, separately for each score.
4. **Tie-breaking rule.** State whether ties use input order, deterministic hash ordering, repeated random ordering, or expected metrics over tie orders. For deterministic or repeated hashes, disclose the inputs, seed policy, and collision fallback.
5. **Permutation test.** Permute each fixed candidate row and verify that an implementation claimed to be independent of input position returns the same ranking or the same expected metric.
6. **Sensitivity analysis.** When ties can affect top-k membership or the positions of relevant items within the top k, report the metric under another tie-breaking rule. Alternatively, report its exact expectation under uniform random tie-breaking. For repeated random tie-breaking, also report variation across runs.

A clean way to remove the systematic relevant-item-first advantage is to permute the entire candidate row independently of relevance before applying a stable sort. This makes the relevant item's position within its tie block uniform in expectation. Each permutation is one random realization, not a fixed invariant ranking. Moving only the relevant item to a randomly selected row position while leaving the negatives in their original order does not generally make its position within the tie block uniform. Full-catalog evaluation can remove the specific relevant-item-first sampled-row construction when catalog order is independent of relevance, but it does not eliminate exact-score ties. A stable sort can still use catalog or storage order as a secondary ranking rule, and a tie crossing the cutoff can still change membership or rank.

7. Limitations

The empirical audits cover two public datasets, rows with one relevant item, and 30 sampled negatives. They demonstrate the mechanism and its magnitude for discrete attribute-based scores, but they do not estimate how often the problem occurs across recommender benchmarks or production systems. Continuous rankers may have few exact ties, while quantization, coarse features, fallback scores, or count-based components may create many.

The Amazon attributes are lexical metadata proxies, and the score families are evaluated as components rather than as complete production recommenders. The purpose is to isolate evaluator behavior, not to compare recommendation algorithms. The residualized Amazon score and MovieLens popularity

were included to show what happens when tie-breaking rarely changes the relevant item's rank. They serve this comparison purpose but are constructed differently.

Full-catalog evaluation is not examined empirically here. In a fixed full catalog, equal scores still require an explicit ordering or averaging rule. The MovieLens results are limited to the aggregate comparison reported in the accepted version because the corresponding row-level candidate and score arrays were not retained.

8. Conclusion

Exact-score ties leave part of a ranking unresolved. If an evaluator resolves them with candidate input order, row construction can enter the metric as an undeclared ranking feature. Holding candidates and scores fixed reveals the effect directly: tie-heavy Amazon and MovieLens scores change by roughly 0.55–0.68 NDCG@10 under deterministic hash tie-breaking, while the residualized Amazon score and MovieLens popularity are nearly unchanged. Deterministic hash ordering provides repeatability; analytic or repeated random tie-breaking describes the average over unresolved orders. Reporting candidate construction, tie prevalence, tie-breaking, and a permutation check makes the evaluated quantity clear and reproducible.

Declaration on Generative AI

During the preparation of this work, the authors used OpenAI ChatGPT and Anthropic Claude for grammar and spelling checks, paraphrasing, structural suggestions, LaTeX and CEURART formatting, and implementation assistance in drafting, debugging, and documenting portions of the experimental and analysis code. The research questions, methodology, experimental decisions, result interpretation, and conclusions were determined by the authors. The authors reviewed and edited all affected material, ran and inspected the experiments, verified the text, calculations, citations, tables, and code, and take full responsibility for the publication's content.

References

[1] J. L. Herlocker, J. A. Konstan, L. G. Terveen, J. T. Riedl, Evaluating collaborative filtering recommender systems, ACM Transactions on Information Systems 22 (2004) 5–53. doi:10.1145/963770.963772.

[2] P. Cremonesi, Y. Koren, R. Turrin, Performance of recommender algorithms on top-n recommendation tasks, in: Proceedings of the Fourth ACM Conference on Recommender Systems, 2010, pp. 39–46. doi:10.1145/1864708.1864721.

[3] R. Cañamares, P. Castells, A. Moffat, Offline evaluation options for recommender systems, Information Retrieval Journal 23 (2020) 387–410. doi:10.1007/s10791-020-09371-3.

[4] R. Cañamares, P. Castells, On target item sampling in offline recommender system evaluation, in: Proceedings of the 14th ACM Conference on Recommender Systems, 2020, pp. 259–268. doi:10.1145/3383313.3412259.

[5] E. Zangerle, C. Bauer, Evaluating recommender systems: Survey and framework, ACM Computing Surveys 55 (2022) 170:1–170:38. doi:10.1145/3556536.

[6] F. McSherry, M. Najork, Computing information retrieval performance measures efficiently in the presence of tied scores, in: Advances in Information Retrieval: 30th European Conference on IR Research, 2008, pp. 414–421. doi:10.1007/978-3-540-78646-7_38.

[7] G. Cabanac, G. Hubert, M. Boughanem, C. Chrisment, Tie-breaking bias: Effect of an uncontrolled parameter on information retrieval evaluation, in: Multilingual and Multimodal Information Access Evaluation, volume 6360 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 112–123. doi:10.1007/978-3-642-15998-5_13.

[8] A. Bellogín, I. Cantador, F. Díez, P. Castells, E. Chavarriaga, An empirical comparison of social, collaborative filtering, and hybrid recommenders, ACM Transactions on Intelligent Systems and Technology 4 (2013) 14:1–14:29. doi:10.1145/2414425.2414439.

[9] J. Lin, P. Yang, The impact of score ties on repeatability in document ranking, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 1125–1128. doi:10.1145/3331184.3331339.

[10] W. Krichene, S. Rendle, On sampled metrics for item recommendation, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 1748–1757. doi:10.1145/3394486.3403226.

[11] W.-S. Chao, Z. Zheng, H. Zhu, H. Liu, Make large language model a better ranker, in: Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, 2024, pp. 918–929. doi:10.18653/v1/2024.findings-emnlp.51.

[12] E. Bito, Y. Ren, E. He, One pass, any order: Position-invariant listwise reranking for llm-based recommendation, in: Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2026, pp. 3625–3629. doi:10.1145/3805712.3809952.

[13] T. Joachims, A. Swaminathan, T. Schnabel, Unbiased learning-to-rank with biased feedback, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, 2017, pp. 781–789. doi:10.1145/3018661.3018699.

[14] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, X. He, Bias and debias in recommender system: A survey and future directions, ACM Transactions on Information Systems 41 (2023) 67:1–67:39. doi:10.1145/3564284.

[15] A. Dzhoha, A. Kurennoy, V. Vlasov, M. Celikik, Reducing popularity influence by addressing position bias, in: Proceedings of the Workshop on Design, Evaluation, and Deployment of Robust Recommender Systems, volume 3924 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024, pp. 17–23. URL: https://ceur-ws.org/Vol-3924/short4.pdf.

[16] Y. Hou, J. Li, Z. He, A. Yan, X. Chen, J. McAuley, Bridging language and items for retrieval and recommendation, arXiv preprint arXiv:2403.03952 (2024). doi:10.48550/arXiv.2403.03952.

[17] F. M. Harper, J. A. Konstan, The MovieLens datasets: History and context, ACM Transactions on Interactive Intelligent Systems 5 (2015) 19:1–19:19. doi:10.1145/2827872.

[18] J. Vig, S. Sen, J. Riedl, The tag genome: Encoding community knowledge to support novel interaction, ACM Transactions on Interactive Intelligent Systems 2 (2012) 13:1–13:44. doi:10.1145/2362394.2362395.