

Tied group sizes make GroupKFold folds depend on the scikit-learn version and the CPU

Tejasvi Singh Tomar

ORCID: <https://orcid.org/0000-0003-4668-3639>

23 October 2026

Abstract

GroupKFold, the grouped K -fold splitter of scikit-learn, sorts groups by size and deals them to folds. Up to version 1.8 it used numpy’s default argsort, which does not fix the order of equal elements and whose SIMD kernels numpy chooses at run time from the CPU. Whenever two groups had the same size, the folds could therefore depend on the numpy version and the CPU. scikit-learn 1.9.0 (June 2026) sorts stably, which removes the CPU dependence but changes such folds relative to all earlier releases. The problem was reported and fixed upstream. We document its consequences. In one research project, 79 of 80 GroupKFold call sites had tied group sizes, and the non-stable orders moved 16 to 79% of the items to another fold. About 60 of the 188 fold-dependent claims of a paper draft changed at printed precision, one comparison reversed, and a pre-registered decision passed under one tie order (Holm-adjusted $p = 0.022$) and failed under another ($p = 0.120$). A self-contained example on the digits data bundled with scikit-learn reproduces the effect. In a pre-registered random sample of public code, 92.8% of GroupKFold uses (95% interval 86.8 to 98.8%) run without shuffling on groups whose sizes can tie. All 8 uses whose data we could rebuild had tied sizes, and in 5 the folds of versions 1.8 and 1.9 differ. We recommend storing folds as data, reporting library versions and checking decisions over several tie orders, and give a helper with a documented tie-break.

1 Introduction

Grouped cross-validation keeps all samples of a group in one fold, so that no model is tested on a group it was trained on. A group can be a subject, the augmented copies of an image, or a block of a time series or a genome, and ignoring such groups inflates performance estimates [1, 2]. In scikit-learn [3] the standard grouped splitter is GroupKFold. By default it takes no random seed, and recent work describes its splits as deterministic, with nothing to seed [4].

Within one environment this holds. Across environments it did not. GroupKFold orders the groups by size, and up to scikit-learn 1.8 the order among groups of equal size was left to numpy’s default sort [5], which does not guarantee any order for equal elements [6]. scikit-learn 1.9.0 sorts stably [7]. Its folds are the same on every machine, but they differ from the folds of all earlier releases whenever group sizes tie. The CPU dependence of numpy’s sort was reported upstream in 2023 and its effect on GroupKFold in 2024 (Section 7). We describe it, show that ties are common in one research project and in public code and papers, and document its consequences.

Table 1: The group order of `GroupKFold` with `shuffle=False`, where `n` holds the group sizes. We read the call in 20 releases from 0.17 (then `LabelKFold`) to 1.8.0, and in 1.9.0 and 1.9.1.

Releases	Sorting call	Order of tied groups depends on
0.17 to 1.8.0	<code>np.argsort(n)[::-1]</code>	numpy version and CPU
1.9.0 and later	<code>np.argsort(n, kind="stable")[::-1]</code>	group labels only

2 Mechanism

2.1 How GroupKFold deals groups

With K folds and the default `shuffle=False`, `GroupKFold` numbers the groups in the sorted order of their labels and counts their samples. It visits the groups from the largest to the smallest and puts each into the fold that holds the fewest samples so far, taking the lowest fold number among equally full folds. Fold k tests on the samples of the groups put into it.

Only the order among groups of the same size is free. If all sizes differ, every release gives the same folds. If two tied groups are visited one after the other, exchanging them exchanges their folds (or changes nothing when both go to the same fold) and leaves the fold sizes, and so the rest of the deal, unchanged. A different order of tied groups therefore moves whole groups between folds.

2.2 Which order each release uses

Table 1 gives the sorting call by release. The ascending `argsort` is reversed to put large groups first, so the stable sort of 1.9 visits tied groups in decreasing label order.

2.3 Why the non-stable order depends on the CPU

numpy guarantees the order of equal values only for stable sorts [6]. Since numpy 1.25, `argsort` of 32- and 64-bit values runs through AVX-512 kernels on CPUs that support them [8], and the numpy 2.0 release notes warn that sorts that are not stable “may return slightly different results in 2.0.0 compared to 1.26.x” [9]. numpy chooses its kernels at run time from the CPU’s features, so one installation can order ties differently on two machines. To find the orders that real installations produce, we ran numpy 1.26.4, 2.0.2, 2.2.6, 2.4.6 and 2.5.4 on one AVX-512 CPU at three dispatch levels: complete, limited to AVX2, and limited to the instructions below AVX2.¹ Each configuration was applied to all 135 group-size vectors that reach `GroupKFold` in the analyses of Sections 3 and 5. The 15 configurations produce exactly four orders: numpy 2 with AVX-512, numpy 2 with AVX2 (the same order for 2.0 to 2.2 and for 2.4 to 2.5), numpy 1.26 with AVX-512, and an order shared by numpy 1.26 with AVX2 and by every version without AVX2. numpy 2.1 and 2.3 were not tested. Together with the stable order, these give five different partitions at 111 of the 134 vectors with tied sizes. A stable sort has only one possible result.

2.4 What the stable order fixes, and what it does not

A stable sort makes the folds the same on every machine. Three caveats remain. First, the folds of 1.9 differ from those of all earlier releases whenever sizes tie, so results computed before an upgrade

¹Through the environment variable `NPY_ENABLE_CPU_FEATURES`: single features up to AVX2, or up to SSE4.2, before numpy 2.4, and the levels `X86_V3` and `X86_V2` from 2.4 on, when x86 targets were grouped by level [10]. From 2.4 on, a list of single features up to AVX2 enables only `X86_V2`. We did not run on a CPU without AVX-512.

cannot be reproduced after it without the old fold assignment. The 1.9.0 changelog calls the change an enhancement that “ensures that the splits are consistent across runs”, and its list of changed models does not include it [7], although the list for 1.0 did name a change to stratified splits [21]. Second, ties are broken by label order, so the folds depend on how the groups are named. In the example of Section 5.3, writing the image identifiers as strings instead of integers moves 77% of the samples to another fold. Third, shuffling the rows before calling `GroupKFold` changes nothing, because groups are indexed by their sorted labels (the example confirms this for both releases). A stable order is reproducible, but it is not more representative than any other order of the tied groups.

3 How often groups tie

Many designs give every group the same size: a fixed number of trials, images or augmented copies per subject or source, blocks of fixed length (all but the last), or matched pairs. When sizes vary, ties remain likely. If G groups take only R distinct sizes, at least $G - R$ of them share their size with another group. Even 50 sizes drawn independently and uniformly from 300 values contain a tie with probability 0.99, and less uniform distributions make ties more likely.

In the audited research project, which benchmarks genomic language models with groups such as pairs of gene blocks and contiguous genomic blocks, 79 of the 80 `GroupKFold` call sites behind the numbers of one paper draft had tied sizes. At these sites 46 to 100% of the groups, out of 45 to 1,275 per site, shared their size with another group. At all 79, each of the four orders of Section 2.3 gave a partition different from the stable one and moved 16 to 79% of the items to another fold. The site without ties had 27 groups of distinct sizes [11].

4 Ties in public code and papers

Beyond one project, we estimated how often public code and papers apply `GroupKFold` to groups whose sizes can tie, under a protocol written before any sample was drawn, with deviations logged. GitHub’s code search lists about 97,800 Python files and 17,400 notebooks that contain the word. We cut the range of file sizes into 20,000 intervals, drew 60 and 100 of them at random and kept every file in them: 438 files, 200 distinct, of which 87 apply `GroupKFold` to their own data (most others are copies of scikit-learn or use other splitters). For each use we followed the groups to their definition and classed their sizes as equal by construction (a fixed number of trials, crops or rows per group), natural units of varying size (subjects, patients, sites, documents), or unknown. For papers, we took the records that mention `GroupKFold` in Europe PMC (335) and OpenAlex (1,323) in random order until we had 40 open full texts from each, so the paper estimates refer to open texts. Of these 80 papers, 66 use it.

Nearly all code uses keep the default `shuffle=False` on groups not built to have distinct sizes (Table 2). Only 2 of the 87 build equal sizes, but natural groups tie as well. For 8 natural-group uses, the repository ships the data from which the groups can be rebuilt. In all 8, at least two groups share a size (29 to 100% of the groups do), and in 5 the folds of scikit-learn 1.8 (with numpy 2.5.4 on an AVX-512 CPU) and 1.9 differ, with 52 to 79% of the samples in another fold. Of the other three, two put each group into its own fold, and in one, with 15 groups, numpy’s default sort happened to return the stable order. Equal sizes by design are more common in papers, which often fix the number of trials per participant. Few uses record what produced their folds, and only one paper shares them (1.5%, 0.3 to 8.1%).

Table 2: GroupKFold in public code and papers. Code: the Python and notebook frames combined with weights 0.85 and 0.15 (normal 95% interval). Papers: Wilson 95% intervals.

	Code (87 uses)	Papers (66)
Equal group sizes by construction	2 (2.0%, 0.0–5.0)	9 (13.6%, 7.3–23.9)
Natural groups of varying size	82	51 (77.3%, 65.8–85.7)
Unknown	3	6
<code>shuffle=False</code> with equal or natural groups	92.8%, 86.8–98.8	shuffling mentioned by 1
scikit-learn version pinned or reported	24 (27.6%, 19.3–37.8)	9 (13.6%, 7.3–23.9)

One sampled repository had met the problem itself. In a study of audio language models on clinical speech, the same code, clips and speaker labels gave different **GroupKFold** splits on a Mac and on Linux GPU servers. The repository releases both fold files and reports that per-layer AUROC curves refitted on the Mac split miss those of the server runs by 0.019 to 0.064 [20]. On its 468 clips from 228 speakers, 335 clips change fold between the two files.

Scaled to the frames, about 26,000 public files (95% interval 10,000 to 47,000) apply **GroupKFold** to their own data. Only 44% of the sampled uses come from repositories last pushed before scikit-learn 1.9.0 appeared; the others may already run with the stable order.

5 Consequences

We report two analyses of the project, rerun with only the tie order changed, and a public example.

5.1 A pre-registered decision

A pre-registered analysis asked whether any of 11 candidate classifiers beats an out-of-fold position baseline, a logistic model fitted with 5-fold **GroupKFold** over pairs of gene blocks, on 11 held-out protein-interaction data sets [13]. Averaged over the data sets, a candidate must score a higher AUROC than the baseline and must raise the AUROC when added to the baseline model, with the larger of the two one-sided bootstrap p -values below 0.05 after Holm adjustment over the 11 candidates. The analysis ran with scikit-learn 1.7.2, and re-scoring with numpy’s default sort on an AVX-512 CPU reproduces it exactly. One candidate passed (Holm-adjusted $p = 0.0221$). With the stable order of scikit-learn 1.9.1 none passed: the lowest Holm-adjusted p was 0.0993, and the former winner had 0.1204, although both of its intervals stayed above zero. The baseline’s mean AUROC moved from 0.4872 to 0.4807, and on one of the three small data sets from 0.386 to 0.499. The other orders of Section 2.3, each rerun with an explicit order, change the outcome again. With numpy 2 on AVX2 the same candidate passes (0.0331). With numpy 1.26 on AVX-512 it passes together with a second candidate (0.0110 and 0.0401). With numpy 1.26 on AVX2 only the second candidate passes (0.0441), and the former winner fails (0.0702). Some candidate thus passes under every non-stable order and none under the stable one. With the homology filter applied in both directions, the former winner passes under the logged order and with numpy 2 on AVX2 (0.0441 and 0.0221), and no candidate passes under the other three orders (lowest Holm-adjusted p 0.1103 to 0.2758). The analysis reports the pass as not robust.

Table 3: The digits example (1,797 groups of one to three samples, five folds). Moved: share of samples in another fold than with scikit-learn 1.9.1, after matching fold labels. Errors: misclassified samples out of 5,149.

Partition	numpy dispatch	Moved	Errors	Accuracy
scikit-learn 1.9.1 (stable order)	any of the three	0	407	0.9210
scikit-learn 1.8.0	AVX-512	0.761	408	0.9208
scikit-learn 1.8.0	X86_V3 (AVX2)	0.754	413	0.9198
scikit-learn 1.8.0	X86_V2 (baseline)	0.733	402	0.9219
50 random tie orders		0.764–0.785	397–431	0.9163–0.9229

5.2 The claims of a paper draft

The programs behind the paper draft had run with scikit-learn 1.9.1, so its numbers rest on the stable order, and stable reruns reproduced all 188 fold-dependent claims at printed precision. Every program was also rerun under the four orders of Section 2.3. Under each, about 60 claims changed at printed precision. Printed out-of-fold baselines moved by up to 0.036 AUROC, and printed gains over position by up to 0.003 [11, 12].

One comparison reversed. The draft places a baseline built from distance and annotation, with 0.706 AUROC on a causal eQTL task, within the range of fine-tuned DNA language models, 0.697 to 0.732. Under the four orders, in the sequence of Section 2.3, the baseline scores 0.694, 0.694, 0.691 and 0.702, so the statement fails under three of them. Over 50 random orders of the tied blocks it scores 0.686 to 0.704 (mean 0.695) and falls below 0.697 in 29. A statement whose interval bound lies within about 0.0001 of zero, that no gain on a set of operon labels has an interval above zero, holds under all four non-stable orders but fails under 3 of 20 random ones [11]. Nor is the stable order a neutral reference. The draft’s 0.706 is higher than the value under each of the 54 other orders tried. In the example below, the accuracy of the stable partition falls within the range of 50 random orders with one grouping and below all of them with the other.

5.3 A public example

The script `demo_ties.py` uses the 1,797 digit images bundled with scikit-learn and the one-pixel shifts (up, left, right and down) of scikit-learn’s example on restricted Boltzmann machines [14]. We keep a shifted copy only if no ink leaves the 8×8 frame, so an image contributes one to three samples, 5,149 in all. The versions of an image must share a fold, so the 1,797 images are the groups, and every group shares its size with others. We computed the partitions with scikit-learn 1.8.0 and 1.9.1 under numpy 2.5.4 on one AVX-512 CPU, with numpy’s dispatch complete, limited to X86_V3 (AVX2) or limited to the X86_V2 baseline, and scored every partition in one process with a logistic regression on the pixels (Table 3).

scikit-learn 1.9.1 gives one partition under all three settings, and 1.8.0 gives a different one under each, with 73 to 76% of the samples in another fold. The out-of-fold accuracy of these four partitions ranges from 0.9198 to 0.9219, and over 50 random tie orders from 0.9163 to 0.9229. When all five versions of every image are kept, as in the original example, all groups have size 5. numpy’s AVX-512 and X86_V3 kernels then return the tied groups in input order, so 1.8.0 agrees with 1.9.1 under those settings and differs only with the baseline sort (64% of the samples moved, accuracy 0.7907 against 0.7885). Whether two environments agree thus also depends on the data. Scoring all partitions took 97 s on one core.

Listing 1: Group folds with a documented tie-break (from `group_folds.py`).

```
import hashlib, numpy as np
def group_folds(groups, n_splits=5, seed=0):
    """Fold of every sample: GroupKFold's deal, ties ordered by SHA-256 of '<seed>:<label>'."""
    labels, idx = np.unique(np.asarray(groups).astype(str), return_inverse=True)
    idx = idx.ravel()
    sizes = np.bincount(idx)
    key = [hashlib.sha256(f"{seed}:{g}".encode()).hexdigest() for g in labels]
    order = sorted(range(len(labels)), key=lambda i: (-sizes[i], key[i]))
    load = np.zeros(n_splits, dtype=np.int64)
    fold_of_group = np.empty(len(labels), dtype=np.int64)
    for i in order:
        # largest group first, into the lightest fold
        f = int(np.argmin(load)) # lowest fold number among equally light folds
        fold_of_group[i] = f
        load[f] += sizes[i]
    return fold_of_group[idx] # e.g. PredefinedSplit(group_folds(groups, 5, seed))
```

6 Recommendations

1. **Store the folds as data.** Save the fold of every sample or group, with a hash, next to the data, and let every analysis read that file. Out-of-fold numbers then reproduce with any library version on any machine [11].
2. **Report what produced the folds.** Record the numpy and scikit-learn versions and, for scikit-learn 1.8 or earlier, the SIMD extensions that numpy uses (`numpy.show_runtime()` lists them). For these releases a version pin is not enough.
3. **Choose the tie-break explicitly.** Listing 1 keeps the deal of `GroupKFold` and orders tied groups by a hash of a seed and the label, so the folds depend only on the labels, their sizes and the seed. It gives the same partition under the three dispatch settings of Section 5.3, after shuffling the rows and with labels written as strings, equals `GroupKFold` when no sizes tie, and deals 100,000 groups in 0.5 s. From release 1.6, `GroupKFold(shuffle=True, random_state=...)` also avoids the CPU dependence, but it balances numbers of groups rather than of samples.
4. **Check decisions over several tie orders.** The data split is a large source of variance in method comparisons [19], and the tie order is part of it. Recompute any decision that rests on out-of-fold numbers over several seeds, and report the spread or the share of seeds under which it holds. For the eQTL baseline of Section 5.2 this reads “0.69 to 0.71, depending on how tied blocks fall into folds”. A pre-registration can fix the seeds and the rule that combines them.
5. **Treat an upgrade to scikit-learn 1.9 as a change of folds.** Results computed with 1.8 or earlier on data with tied group sizes will not reproduce with 1.9. Keep the old fold files, or recompute the folds with the old versions on a CPU with the same SIMD extensions.

7 Prior reports and what this note adds

The earliest report we found of `argsort`’s tie order changing with the CPU is numpy issue #24064 (27 June 2023). After the AVX-512 `argsort` of numpy 1.25, tests failed on GitHub’s Ubuntu runners because equal elements came back in another order, and the reporter closed it on finding `kind="stable"` [8, 22]. For `GroupKFold`, the earliest is scikit-learn pull request #28464 (19 February 2024, GitHub user marikabergengren), which proposed a stable sort “to produce consistent splits

across installs” [15]. Issue #29495 (15 July 2024, user blitchj) showed different partitions on two machines with numpy 1.26.4 and scikit-learn 1.5.1, traced them to ties in `np.argsort` and came with pull request #29496 [16]. A comment there (17 November 2024, user VladKha) reported different results between CPU and GPU instances of Kaggle Notebooks and noted that the documentation did not mention the behaviour. blitchj also reported the analogous problem in recursive feature elimination (issue #29531, fixed by pull request #29532 [17]), where a maintainer replied that “in general, we do not guarantee the same results between machines”. Pull request #28464 was merged on 6 April 2026 and released in 1.9.0 [7]. In the literature, Zhang [18] pins scikit-learn 1.8.0 because “GroupKFold split assignment varies across scikit-learn releases”, which moved a reported mean by up to 0.01, and Ben-Ami et al. [4] describe the splits as deterministic. Neither mentions the CPU.

Nearby effects are documented. Hardware-dependent data-loader settings create performance differences of up to 4% between identical DNA language models on one benchmark and change their ranking [23]. The rule that breaks ties between scores changes reported gains and rankings in group recommendation [24]. In a survey, 83.8% of 901 researchers and practitioners were unaware of or unsure about implementation-level variance in deep learning [25]. Prevalence studies have counted papers that ignore groups (28 of 62 reviewed studies that predict clinical outcomes from wearable sensors or smartphones used record-wise cross-validation [26]) and notebooks with data leakage (29.6% of 107,603 on GitHub [27]).

The mechanism and its fix belong to the reports above. Searching the scikit-learn and numpy issue trackers, the scikit-learn changelogs, arXiv and the web on 22 October 2026, we found no measurement of how often equal group sizes occur in practice or of how far the tie order moves reported results. This note adds the order across scikit-learn 0.17 to 1.9 and numpy 1.26 to 2.5 (including the change of folds at 1.9), counts from the 80 call sites of one project, two consequences there, a pre-registered estimate for public code and papers, and a public example with a helper. The scripts `tie_odds.py`, `numpy_order_census.py`, `demo_ties.py` and `group_folds.py` and their logs, and the protocol, samples, classifications and scripts behind Section 4, are available from the author on request. The project analyses are documented in unpublished reports [11–13].

Acknowledgements

Generative AI tools were used to write code, run analyses and draft text; the author directed the work and takes full responsibility for its content.

References

- [1] D. R. Roberts, V. Bahn, S. Ciuti, et al. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 40:913–929, 2017.
- [2] S. Kapoor and A. Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4:100804, 2023.
- [3] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [4] D. Ben-Ami, K. Cohen, and C. Baskin. Have I seen enough? Frozen video-language models encode evidence readiness. arXiv:2610.08560v2, 2026.
- [5] C. R. Harris, K. J. Millman, S. J. van der Walt, et al. Array programming with NumPy. *Nature*, 585:357–362, 2020.
- [6] NumPy developers. `numpy.argsort`, NumPy reference. <https://numpy.org/doc/stable/reference/generated/numpy.argsort.html>.
- [7] scikit-learn developers. Release history, version 1.9 (1.9.0, June 2026; 1.9.1, September 2026). https://scikit-learn.org/stable/whats_new/v1.9.html.

- [8] NumPy developers. NumPy 1.25.0 release notes: faster np.argsort on AVX-512 enabled processors. <https://numpy.org/doc/stable/release/1.25.0-notes.html>.
- [9] NumPy developers. NumPy 2.0.0 release notes: minor changes in behavior of sorting functions. <https://numpy.org/doc/stable/release/2.0.0-notes.html>.
- [10] NumPy developers. NumPy 2.4.0 release notes: modulate dispatched x86 CPU features. <https://numpy.org/doc/stable/release/2.4.0-notes.html>.
- [11] GroupKfold tie-order audit of a paper draft. Unpublished report, 21 October 2026.
- [12] Tie-order follow-up: the interaction, eQTL, BEND, operon and pre-registered analyses rerun under each order. Unpublished report, 22 October 2026.
- [13] Pre-registered analysis of classifiers trained on layout-free interaction labels. Unpublished report, October 2026.
- [14] scikit-learn developers. Restricted Boltzmann Machine features for digit classification. Example `plot_rbm_logistic_classification.py`, function `nudge_dataset`.
- [15] scikit-learn pull request #28464, opened 19 February 2024, merged 6 April 2026. <https://github.com/scikit-learn/scikit-learn/pull/28464>.
- [16] scikit-learn issue #29495, opened 15 July 2024, and pull request #29496. <https://github.com/scikit-learn/scikit-learn/issues/29495>.
- [17] scikit-learn issue #29531 and pull request #29532, merged 9 April 2026. <https://github.com/scikit-learn/scikit-learn/issues/29531>.
- [18] G. Zhang. The visual insensitivity gap: Diagnosing when vision-language models fail to use visual evidence. arXiv:2609.00868, 2026.
- [19] X. Bouthillier, P. Delaunay, M. Bronzi, et al. Accounting for variance in machine learning benchmarks. In *Proceedings of Machine Learning and Systems 3 (MLSys 2021)*, 2021.
- [20] GitHub repository cparw/audio-llm-utility-gap, `folders/README.md` at commit 729bb85a, accessed 22 October 2026. <https://github.com/cparw/audio-llm-utility-gap>.
- [21] scikit-learn developers. Release history, version 1.0, “Changed models”. https://scikit-learn.org/stable/whats_new/v1.0.html.
- [22] numpy issue #24064, “BUG: Argsort can produce different output on different architectures”, opened and closed 27 June 2023. <https://github.com/numpy/numpy/issues/24064>.
- [23] D. Greco and K. Rawlik. Same model, better performance: the impact of shuffling on DNA language models benchmarking. arXiv:2510.12617, 2025.
- [24] S. D. Ma and P. J. Cheng. Are we really making progress in group recommendation? Unmasking the tie-breaking illusion. In *Proceedings of the 20th ACM Conference on Recommender Systems*, pages 861–869, 2026.
- [25] H. V. Pham, S. Qian, J. Wang, et al. Problems and opportunities in training deep learning software systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 771–783, 2020.
- [26] S. Saeb, L. Lonini, A. Jayaraman, D. C. Mohr, and K. P. Kording. The need to approximate the use-case in clinical machine learning. *GigaScience*, 6:gix019, 2017.
- [27] C. Yang, R. A. Brower-Sinning, G. A. Lewis, and C. Kästner. Data leakage in notebooks: static detection and better processes. In *Proceedings of ASE ’22*, 2022. arXiv:2209.03345.