Efficient cross-validation with groups and stratification

Anonymous Author
Anonymous Institution

Abstract

Cross-validation is used throughout machine learning for model training and evaluation. In classification data sets, stratification is used to ensure that splits preserve class proportions. In structured data (image segmentation, medical visits, etc.), grouping is used to prevent data leakage (data from the same group should not appear in both train and test sets). Fold assignment is trivial with either stratification or grouping, but it is NP-hard with both stratification and grouping. Scikit-learn implements a heuristic that obtains reasonable fold assignments in this case, but its time and memory complexity can be prohibitive for large data. We propose a new heuristic algorithm with asymptotically less time and memory usage, that can be used on large data sets. In an empirical analysis of four real data sets, the proposed algorithm provides comparable fold assignments, and is asymptotically about ten times faster than the previous algorithm.

1 INTRODUCTION

Many types of modern artificially intelligent systems are powered by machine learning: image classification, medical diagnosis/prognosis, automatic translation, etc. K-fold cross-validation [Arlot and Celisse, 2010, Raschka, 2020] is a fundamental data-splitting technique that is used for both training (tuning hyperparameters of a learning algorithm) and evaluation (comparing prediction accuracy of different learning algorithms). In a data set of N observations (rows of tabular data), K-fold cross-validation involves assigning a fold ID (integer between 1 and K) to each of the N rows. For each fold ID from 1 to K, the corresponding rows form a test set, and the other rows form a train set (for the case of evaluation). Learning algorithms are fit to each train set, the resulting K test error values are used to estimate how the learning algorithm would perform on new data.

Preliminary work. Under review by AISTATS 2027. Do not distribute.

1.1 Issues with unstructured cross-validation

This paper proposes a new algorithm for cross-validation in machine learning data sets with two kinds of structure: groups and strata. Groups are rows that should stay together when splitting, to avoid data leakage, which can cause over-optimistic estimates of prediction error rates [Kaufman et al., 2012, Kapoor and Narayanan, 2023]. Examples of groups include medical records for the same patient (in diagnostic/prognostic prediction problems), and pixels labeled from the same polygon (in image segmentation problems). Strata are classes of rows for which the relative proportions should be preserved when splitting. Examples of strata include the output label in classification problems, such as medical diagnosis or image segmentation [Russell et al., 2008, Asif et al., 2025]. To understand why our proposed algorithm is useful, we first discuss issues with unstructured cross-validation in structured data.

1.1.1 Neither stratification nor groups

Because the goal of data-splitting is to estimate prediction error in the same distributional setting as the original data set, it is important for the splits to be representative of the original data. In an unstructured data set, we would at least like to ensure that each test set has the same number of rows. For example, consider the AZtrees data [Hocking et al., 2026], which represent an image segmentation problem. There are $N = 5956$ rows, each representing a pixel in an image that has been labeled as containing a tree (positive class) or not (background negative class), so in $K = 3$ fold cross-validation, we expect each fold to have approximately $N/K = 1985.33$ rows. Standard unstructured cross-validation repeats the sequence of fold values $(1, \ldots, K, 1, \ldots, K, \ldots)$, to obtain an N-vector which is then randomly shuffled. This guaran-

Case	Structure (algorithm)	Class y =stratum	data rows in folds f1	f2	f3	groups in one fold (ok) f1	f2	f3	groups in multiple folds (leakage) f1–2	f2–3	f1,3	f1–3
A)	no group no strata (trivial)	Negative	1749	1761	1772	1		3	4		4	60
		Positive	237	224	213	16	13	10	13	15	8	42
		Total	**1986**	**1985**	**1985**	17	13	13	17	15	12	102
B)	no group yes strata (trivial)	Negative	**1761**	**1761**	**1760**	2	3		1	3	3	60
		Positive	**225**	**225**	**224**	14	11	12	14	13	9	44
		Total	**1986**	**1986**	**1984**	16	14	12	15	16	12	104
C)	yes group no strata (trivial)	Negative	1806	1066	2410	23	25	24				
		Positive	320	187	167	40	38	39				
		Total	2126	1253	2577	**63**	**63**	**63**				
D)	yes group no strata (proposed)	Negative	1803	1756	1723	19	28	25				
		Positive	183	229	262	39	36	42				
		Total	**1986**	**1985**	**1985**	58	64	67				
E)	yes group yes strata (proposed)	Negative	**1761**	**1761**	**1760**	19	26	27				
		Positive	**225**	**224**	**225**	39	39	39				
		Total	**1986**	**1985**	**1985**	58	65	66				

Table 1: Demonstration of trivial and proposed algorithms for five cases of $K = 3$ fold cross-validation in AZtrees data, with $N = 5956$ rows, $G = 189$ groups, and $S = 2$ strata (same as binary label y). A missing/empty table cell indicates group count=0. The proposed algorithm with stratification and groups (case E) gives nearly equal data counts across folds (**bold**), and no leakage (zero groups in multiple folds).

tees approximately the same number of rows in each fold (at most one row difference, when N/K is not an integer). For the AZtrees example, this results in 1985–1986 rows per fold (Table 1 Case A Total).

1.1.2 Stratification but no groups

However, the random shuffling in unstructured cross-validation can cause substantial variation between class frequencies, which is undesirable. For the AZtrees example, one random shuffle gave 1749–1772 Negative rows and 213–237 Positive rows (Table 1 Case A Negative and Positive). To get class frequencies that are approximately equal across folds (at most one row different), cross-validation can be stratified using the class. For a stratum/class with c rows, we repeat the fold values $(1, \ldots, K, 1, \ldots, K, \ldots)$, to obtain a c-vector, then assign the shuffled fold values to the rows in that class/stratum. For the AZtrees example, stratification results in 1760–1761 negative rows per fold, and 224–225 positive rows per fold. (Table 1 Case B Negative and Positive).

1.1.3 Groups but no stratification

In a structured data set with a grouping variable, the data-splitting technique should work on groups instead of rows, to avoid data leakage [Kaufman et al., 2012, Kapoor and Narayanan, 2023]. For example, medical data are often grouped by patients, who may have a different stratum/class label for each visit/row. Another example is image segmentation (AZtrees data, Table 1), where groups are polygons, and all pixels/rows in the polygon/group are assigned the corresponding class label. We would like to avoid generating a split with some rows of a group assigned to the train set, and other rows from that group assigned to the test set (data leakage), which would result in over-optimistic estimates of test error rates. In the AZtrees data, cross-validation without grouping by polygon results in data leakage: some groups are randomly assigned to multiple folds (Table 1 Cases A and B, 1–104 groups in multiple folds).

To modify cross-validation for a data set with G groups, we repeat the fold values $(1, \ldots, K, 1, \ldots, K, \ldots)$, to obtain a G-vector, which is randomly shuffled to assign each group to a fold. Grouped cross-validation always results in approximately the same number of groups in each fold (max difference of 1 group if G/K is not an integer). The AZtrees data has $G = 189$ groups, so grouped cross-validation with $K = 3$ folds results in $G/K = 63$ groups per fold (Table 1 Case C Total).

However, simple grouped cross-validation only ensures approximately equal numbers of groups (not rows) across folds, so each fold can have very different row counts. In the AZtrees example, one random shuffle gave 1253–2577 rows per fold (Table 1 Case C Total). Furthermore, simple grouped cross-validation does not perform stratification, so each fold can have very different class frequencies. In the AZtrees example, one

Algorithm	Time	Memory
Wasikowski (previous, scikit-learn)	$O(N \log N + K^2SG)$	$O(N + KS + SG)$
Wasikowski-L (proposed)	$O(N \log N + K^2SG)$	$O(N + KS)$
Residual Sum of Squares (RSS, proposed)	$O(N \log N + KSG)$	$O(N + KS)$

Table 2: Proposed algorithms have reduced asymptotic time and memory complexity for computing K-fold cross-validation assignment, given N data rows in S strata and G groups.

random shuffle gave 167–320 positive rows per fold, and 1066–2410 negative rows per fold (Table 1 Case C Negative and Positive).

1.2 Proposed algorithm for cross-validation with stratification and groups

Are there efficient algorithms that can handle stratification and grouping? Yes, but any efficient algorithm must be heuristic, because this assignment problem is NP-hard when each group has an arbitrary number of rows per stratum. An ideal algorithm would result in the same class frequencies in each fold, and no data leakage (Table 1 Case E).

This paper proposes a new heuristic algorithm which performs greedy minimization of the difference between expected and ideal row counts across folds, while respecting the group constraint to avoid data leakage. In the AZtrees data, one random run of the proposed heuristic without stratification gave 1985–1986 rows per fold, 1723–1803 negative labels per fold, 183–262 positive labels per fold, and no data leakage (Table 1 Case D). With stratification, the proposed heuristic gave an ideal result: 1985–1986 rows per fold, 1760–1761 negative labels per fold, 224–225 positive labels per fold, and no data leakage (Table 1 Case E).

Our proposed heuristic is inspired by a Kaggle competitor's algorithm for minimizing variance between row counts in each fold [Wasikowski, 2019]. We refer to this previous algorithm as the Wasikowski heuristic, which is used in the StratifiedGroupKFold class, available in scikit-learn since version 1.0 [Hermida and Martynov, 2020, sklearn developers, 2021]. However, the Wasikowski heuristic is relatively inefficient (Table 2). For K folds, G groups, and S strata, it has a $O(K^2SG)$ term in time complexity, and a $O(SG)$ term in memory complexity. The quadratic time is too slow for large numbers of folds, which are useful when comparing with blocked CV in spatial data [Ploton et al., 2020] ($K \approx 100$ spatial blocks). The memory complexity can be prohibitive if both S and G are large, for example in large language models [Yang, 2024] ($S \approx 10^4$ tokens, $G \approx 10^6$ documents), and other data with many classes like ImageNet [Russakovsky et al., 2015]. This paper proposes new heuristic algorithms with reduced asymptotic time and memory complexity (Table 2).

1.3 Contributions

This paper presents five new research contributions related to cross-validation with groups and strata. After reviewing related research (Section 2), we present a new abstract characterization of the previous Wasikowski algorithm, which allows understanding and improving three sub-routines (Contribution 1, Section 3.1). Next, we propose a new online counting algorithm, with asymptotically less memory usage (Contribution 2, Section 3.3), which permits new applications to data sets with large numbers of strata and groups. We propose a new Residual Sum of Squares (RSS) minimization algorithm which is asymptotically faster than the previous Wasikowski algorithm (Contribution 3, Section 3.4). To demonstrate the advantages of our new algorithm, we propose synthetic data sets to use as test cases (Contribution 4, Section 4). Finally, we present systematic empirical comparisons with 5 baselines, in terms of fold assignment accuracy and computation time in four real data sets (Contribution 5, Section 5).

2 RELATED WORK

A typical application of stratification is in classification problems, where the stratum variable is defined as the label/output variable to predict. Other examples of stratification variables are domain/subset, when evaluating domain adaptation algorithms, and when estimating similarity of data subsets for learning purposes [Hocking et al., 2026]. Assigning groups of data to cross-validation folds is similar to the bin packing problem, which is NP-hard, but with fast heursitic algorithms that have reasonable approximation error bounds [Johnson, 2016]. Two classic algorithms are "first fit decreasing" and "best fit decreasing," both of which sort the list of objects by size, and have asymptotic worst case ratios of 11/9 (amount of more bins used than necessary). Whereas bin packing has a variable number of bins used for N items, cross-validation has a fixed number of K bins, so is a different NP-hard problem: multiway number partitioning, which has been called the "easiest hard problem" [Hayes, 2002]. Cross-validation fold assignment is an instance of multiway number partitioning in which each fold is a bin, and each group is represented by a multi-dimensional

size vector (one element per stratum). Simple heuristics for this problem are similar to bin packing: sort by size and then use greedy assignment to the emptiest bin. Other heuristics for this problem include genetic algorithms and local search Pop and Matei [2013], which are too complex for the cross-validation fold assignment problem. We provide empirical comparisons with previous algorithms and software for cross-validation fold assignment in scikit-learn [Pedregosa et al., 2011], `bioLeak` [Korkmaz, 2026], `rsample` [Frick et al., 2026], and `origami` [Coyle and Hejazi, 2018, Coyle et al., 2026].

3 ALGORITHMS

Assume we have a data set with N rows in S strata and G groups. We want to perform K-fold cross-validation, such that data in each group are assigned the same fold ID (no data leakage), and the strata proportions are approximately equal across folds.

3.1 Pseudo-code for greedy fold assignment

The previous Wasikowski algorithm, and our proposed algorithms, are all greedy heuristics which we summarize using the pseudo-code:

```
def assign_folds(strata, groups, N_folds):
  count_matrix = zeros(N_strata, N_folds)
  assignment = {}
  for group in Sort(groups):
    best_fold = get_best_fold(group)
    assignment[group] = best_fold
    count_matrix[,best_fold] += count(group)
  return assignment
```

This is a novel abstract characterization of the previous algorithm (Contribution 1), which facilitates understanding and improving its computational complexity. The algorithm inputs N-vectors `strata` (S unique values) and `groups` (G unique values), as well as `N_folds` (integer, K). The greedy algorithm requires storing the current number of strata assigned to each fold in `count_matrix`, which starts with zeros. The $O(N \log N)$ time `Sort()` sub-routine is used to determine the order groups will be considered by the greedy algorithm, and then each group is assigned to a fold in the for loop. For each group, the `get_best_fold()` sub-routine determines the fold assignment, which is stored for output, and used to update `count_matrix`.

3.2 Previous Wasikowski algorithm

The previous Wasikowski algorithm uses the following choices for the sub-routines in the pseudo-code. The `Sort()` sub-routine uses the variance of the strata counts (max variance assigned first), which has some drawbacks that are discussed in Section 4. The $O(K^2S)$ time `get_best_fold()` sub-routine loops over possible folds, adding the current group's strata counts to current fold's column of `count_matrix`, which is then summarized by computing standard deviation (SD) over folds for each stratum, and finally mean of the SD values over strata. The `count(group)` sub-routine involves checking a matrix of counts for each stratum and group, which takes $O(SG)$ memory. This algorithm is implemented in scikit-learn [Pedregosa et al., 2011], and has overall $O(N \log N + K^2SG)$ time, $O(N + KS + SG)$ memory complexity (Wasikowski algorithm in Table 2).

3.3 Proposed online counting algorithm for reduced memory complexity

The previous algorithm's $O(SG)$ term in memory complexity can be prohibitive if there are a large number of strata and groups. We propose to sort all data rows by group, so the for loop over groups becomes a for loop over data rows, and the `count(group)` sub-routine computes a vector of strata counts for the current group (no need for a matrix of counts for every group and stratum). This proposed online counting algorithm can be used with the previous heuristic (Wasikowski-L in Table 2), and with our proposed squared error minimization heuristic (RSS in Table 2). Both are the first such algorithms with no $O(SG)$ term in memory complexity (Contribution 2), which permits new applications of these algorithms to data with a large number of strata and groups.

3.4 Proposed RSS minimization for reduced time complexity

The main new idea of the proposed algorithm is to minimize the Residual Sum of Squares (RSS) between the ideal and actual row counts per fold/stratum (Contribution 3).

3.4.1 Proposed Sort() sub-routine

For each row $i \in \{1, \ldots, N\}$, we have a stratum $s_i \in \{1, \ldots, S\}$. The algorithm begins by counting the number of rows in each stratum, then dividing by the number of folds, to obtain the ideal number of rows which could appear in each fold. For each stratum $\sigma \in \{1, \ldots, S\}$, the ideal number of rows is

$$n^*_\sigma = \sum_{i \in \{1,\ldots,N\}} I[s_i = \sigma]/K, \tag{1}$$

where $I[\cdot]$ is the indicator function (1 if argument is true, 0 otherwise). Table 3 (row "ideal per fold") shows

Data:	respiratory rows in		AZtrees rows in		PetAdoption rows in					Laribi2024 rows in	
Algo. fold	s1	s2	s1	s2	s1	s2	s3	s4	s5	s1	s2
ideal per fold	49	62	168.5	1320.5	102.5	772.5	814.75	1009.25	1049.25	9176.25	52945
RSS 1	48	64	168	1478	102	772	815	1009	1050	9177	52945
proposed 2	50	62	169	1268	103	772	814	1010	1049	9176	52945
3	51	61	169	1268	103	773	815	1009	1049	9176	52945
4	47	61	168	1268	102	773	815	1009	1049	9176	52945
Wasikowski 1	49	63	160	1478	103	773	815	1010	1047	9176	52946
previous 2	49	59	172	1268	103	773	814	1009	1050	9176	52945
3	50	62	171	1268	102	772	815	1009	1050	9177	52945
4	48	64	171	1268	102	772	815	1009	1050	9176	52944
random 1	52	60	175	1619	106	869	955	1217	1079	9121	53107
baseline 2	35	77	160	794	99	826	922	1079	1153	9263	53186
3	47	65	188	1014	113	738	641	853	985	9111	52800
4	62	46	151	1855	92	657	741	888	980	9210	52687

Table 3: Typical results, $K = 4$ fold cross-validation on four real data sets with groups and stratification. Columns "rows in" show number of rows in each fold and stratum (s1 = stratum 1, etc). Counts closer to ideal are better (red numbers highlight slight differences between algorithms). Overall the proposed RSS and previous Wasikowski algorithms gave similar results (both significantly better than random baseline).

n^*_σ values for $K = 4$ fold cross-validation in four real data sets.

The proposed Sort() sub-routine sorts groups by RSS (increasing). To compute the RSS for each group, we first compute $c_1, \ldots, c_S$, the counts of rows in each stratum for the current group, `count(group)` in the pseudo-code. Then we define the RSS for that group as the squared error between ideal and group counts, summed over strata, $\sum_{\sigma\in\{1,\ldots,S\}}(c_\sigma - n^*_\sigma)^2$. If there are two groups with the same RSS value, we break ties by sorting using the number of rows in that group (decreasing), Nrow $= \sum_{\sigma\in\{1,\ldots,S\}} c_\sigma$, and do a third level of tie-breaking by sorting by the weighted sum (decreasing), Wsum $= \sum_{\sigma\in\{1,\ldots,S\}} c_\sigma n^*_\sigma$. These sorting rules can be justified as analogous to the "pack larger bins first" heuristic [Johnson, 2016]. In Section 4, we propose illustrative synthetic data to justify these sorting rules.

3.4.2 Proposed greedy fold assignment

Our proposed `get_best_fold()` sub-routine is defined as follows. For each stratum σ and fold $\phi \in \{1, \ldots, K\}$, let $\hat{n}_{\sigma,\phi}$ be the number of data with that stratum assigned to that fold (`count_matrix` in the pseudo-code). We propose to find a fold assignment which minimizes the residual sum of squares,

$$\text{RSS} = \sum_{\sigma\in\{1,\ldots,S\}}\sum_{\phi\in\{1,\ldots,K\}} (\hat{n}_{\sigma,\phi} - n^*_\sigma)^2. \quad (2)$$

The algorithm starts with all $\hat{n}_{\sigma,\phi} = 0$ and then assigns a fold to one group at a time. We propose assigning the group to the fold that minimizes the RSS after assignment,

$$\min_{k\in\{1,\ldots,K\}} \sum_{\sigma\in\{1,\ldots,S\}}\sum_{\phi\in\{1,\ldots,K\}} \left((\hat{n}_{\sigma,\phi} + c_\sigma I[\phi = k]) - n^*_\sigma\right)^2. \quad (3)$$

Expanding the square and simplifying yields

$$\min_{k\in\{1,\ldots,K\}} \sum_{\sigma\in\{1,\ldots,S\}}\sum_{\phi\in\{1,\ldots,K\}} \hat{n}^2_{\sigma,\phi} - 2\hat{n}_{\sigma,\phi}n^*_\sigma + {n^*_\sigma}^2 + I[\phi = k]c_\sigma(2\hat{n}_{\sigma,\phi} + c_\sigma - 2n^*_\sigma). \quad (4)$$

The first three terms above are the current RSS (2), which does not depend on k, so it is a constant that we can remove from the min. Furthermore, the indicator function cancels the sum over folds ϕ:

$$\text{RSS} + \min_{k\in\{1,\ldots,K\}} \sum_{\sigma\in\{1,\ldots,S\}} c_\sigma(2\hat{n}_{\sigma,k} + c_\sigma - 2n^*_\sigma), \quad (5)$$

which can be solved in $O(KS)$ time. Taking the argmin, we remove terms that do not depend on k:

$$k^* = \arg\min_{k\in\{1,\ldots,K\}} \sum_{\sigma\in\{1,\ldots,S\}} c_\sigma \hat{n}_{\sigma,k}. \quad (6)$$

We propose using (6) for the `get_best_fold()` sub-routine in the pseudo-code, with tie-breaking by taking the fold which results in minimal counts larger than ideal. The proposed RSS minimization procedure (6) has a sum over S terms inside a min over K terms, yielding a $O(SK)$ time `get_best_fold()` sub-routine. Comparing to the previous Wasikowski algorithm, the proposed RSS minimization is asymptotically faster (linear rather than quadratic in the number of folds K). The overall complexity of the proposed RSS minimization algorithm is therefore $O(N \log N + KSG)$ time and $O(N + KS)$ memory (Table 2).

	Proposed: sort RSS increasing, min RSS						Previous: sort Var decreasing, min Var					
	counts			total counts in			counts			total counts in		
iteration	in group	fold	RSS	fold1	fold2	fold3	in group	fold	Var	fold1	fold2	fold3
1	1,3	1	0	1,3			0,2	1	2.0	0,2		
2	1,2	2	1	1,3	1,2		1,3	2	2.0	0,2	1,3	
3	0,2	3	2	1,3	1,2	0,2	1,2	3	0.5	0,2	1,3	1,2
4	1,1	3	4	1,3	1,2	1,3	0,1	1	0.5	0,3	1,3	1,2
5	0,1	2	5	1,3	1,3	1,3	1,1	1	0.0	1,4	1,3	1,2

Table 4: Demonstration of proposed algorithm with $N = 12$ rows, $G = 5$ groups, $S = 2$ strata, and $K = 3$ folds. Counts columns show numbers of rows per stratum, for example 1,3 means 1 row in first stratum and 3 rows in second stratum. The proposed greedy RSS minimization works to compute an ideal fold assignment (**left**, total counts 1,3 in each fold), but previous variance minimization does not (**right**, 2–4 counts in second stratum).

	Sort by RSS					Proposed: Sort by RSS, -Nrow, -Wsum					
	counts			total counts in		counts				total counts in	
iteration	in group	RSS	fold	fold1	fold2	in group	Nrow	Wsum	fold	fold1	fold2
1	5,3	5	1	5,3		5,3	8	45	1	5,3	
2	1,2	34	2	5,3	1,2	3,0	3	18	2	5,3	3,0
3	1,2	34	2	5,3	2,4	1,2	3	16	2	5,3	4,2
4	1,2	34	2	5,3	3,6	1,2	3	16	2	5,3	5,4
5	3,0	34	2	5,3	6,6	1,2	3	16	1	6,5	5,4
6	1,1	41	1	6,4	6,6	1,1	2	11	2	6,5	6,5

Table 5: Demonstrating proposed tie-breaking algorithm with $N = 22$ rows, $G = 6$ groups, $S = 2$ strata, and $K = 2$ folds. Counts columns show numbers of rows per stratum, for example [5,3] means 5 rows in first stratum and 3 rows in second stratum. **Left:** sorting only by increasing RSS can assign group [3,0] in iteration 5, which results in a non-ideal fold assignment. **Right:** sorting by increasing RSS, then tie-breaking by decreasing Nrow and Wsum, assigns group [3,0] in iteration 2, which results in ideal fold assignment.

4 SYNTHETIC DATA RESULTS

To justify our new algorithm, we propose synthetic data sets to use as test cases (Contribution 4).

4.1 Test case to justify sorting by RSS

We propose the following synthetic test case to show the advantage of the proposed algorithm (sort groups by RSS increasing), relative to the previous Wasikowski algorithm (sort by variance decreasing). Consider the following data set with $S = 2$ strata, $G = 5$ groups, $N = 12$ data, and $K = 3$ fold cross-validation (Table 4). We use the notation $[c_1, c_2]$ for c_1 rows in stratum 1, and c_2 rows in stratum 2. Consistent with the "pack larger bins first" heuristic, the proposed algorithm sorts the five groups in the order [1,3], [1,2], [0,2], [1,1], [0,1]. The previous Wasikowski algorithm sorts in an order that is not consistent with the number of rows: [1,2] and [0,1] both have variance 0.5, so are assigned folds before [1,1] (zero variance). For these data, the proposed RSS minimization always obtains ideal counts per fold, [1,3] (Table 4, left). In contrast, the previous method can result in sub-optimal counts: [1,4],[1,3],[1,2] (Table 4, right).

4.2 Test case to justify tie-breaking

We propose the following synthetic data set where the tie-breaking rules (Section 3.4.1) are necessary for optimal fold assignment. It involves $N = 22$ data with $G = 6$ groups, $S = 2$ strata, and $K = 2$ folds (Table 5). The groups are [1,1], [3,0], [5,3], and three groups with the counts [1,2]. The total is [12,10], so the ideal is $n^* = [6, 5]$ per fold. The groups [3,0] and [1,2] are tied for the first two sorting criteria: RSS=34 and Nrow=3. To third level of tie-breaking should prefer the group [3,0] which has more rows than [1,2] in the larger first stratum. Our proposed third criterion is the weighted sum, Wsum $= \sum_{\sigma \in \{1,\ldots,S\}} c_\sigma n^*_\sigma$, which is 18 for [3,0] and 16 for [1,2]. Using this tie-breaking rule in these data, the greedy heuristic always obtains ideal counts of [6,5] for both folds (Table 5, right). Omitting this rule can result in sub-optimal counts: one fold with [6,4] and the other with [6,6] (Table 5, left).

5 REAL DATA RESULTS

We present systematic comparisons with previous algorithms and software (Contribution 5). All results were computed on a 2025 laptop (Intel Core Ultra 5 125U CPU).



Figure 1: Fold assignment error using six algorithms on four real data sets. Each dot shows one of 10 random seeds, with randomness used to tie-break in sort step. Ideal results have small RMSE values (indicating good agreement with ideal strata counts per fold), and black rather than red dots (indicating no data leakage).

5.1 Data sets

We chose four data sets that represent a variety of machine learning applications, over several orders of magnitude of size, from $N = 444$ to 248,485 rows, $G =$ 111 to 123,646 groups, $S = 2$ to 5 strata (Figure 1).

We considered two medical data sets in which groups are patients and rows are medical visits, so grouped cross-validation is important to ensure accurate estimation of prediction error on new patients. The **respiratory** data have $N = 444$ rows and $G = 111$ groups for respiratory illness prediction, from the geepack package in R [Halekoh et al., 2006]. The **Laribi2024** data from Zenodo [Laribi et al., 2024] have $N = 248,485$ rows and $G = 123,646$ groups for one year mortality prediction at emergency room admission [Laribi et al., 2025]. There are $S = 2$ strata/labels: affected or not by illness/death.

In image segmentation, labels are often created using polygons, and each pixel in the polygon becomes a data row with the corresponding label. Using polygons as groups in cross-validation ensures accurate estimation of prediction error on new polygons. The **AZtrees** data have $N = 5956$ rows and $G = 189$ groups for predicting presence versus absence of trees in satellite images of Arizona [Hocking et al., 2026] ($S = 2$ strata).

The previous Wasikowski algorithm was originally motivated by a Kaggle data set about pet adoption [Wasikowski, 2019]. In the **PetAdoption** data, there are $N = 14,993$ rows, each row is a pet, and pets are grouped by rescuer ($G = 5595$ groups). There are $S = 5$ strata/labels: categories of adoption speed.

5.2 Baseline algorithms

We compared the proposed RSS algorithm to five others. First, the random baseline is the trivial algorithm which assigns each group to a random fold (no leakage, but high error rate, case C in Figure 1). Second, we ran the previous Wasikowski algorithm, which has a Python/numpy implementation in scikit-learn [Pedregosa et al., 2011]. We re-implemented this algorithm in C++ for more controlled timing comparisons with our proposed C++ code for RSS minimization. We also ran three packages in R [R Core Team, 2026] with functions for computing fold assignment using stratification and groups (not re-implemented in C++ because they were not competitive in terms of accuracy): `bioLeak` version 0.3.8 [Korkmaz, 2026], `rsample` version 1.3.2 [Frick et al., 2026], `origami` version 1.0.8 [Coyle and Hejazi, 2018, Coyle et al., 2026].

5.3 Accuracy of fold assignment

To compare fold assignment accuracy, we ran all six algorithms on all four data sets, using $K \in \{2, 4, 8\}$ folds, and ten random seeds which control final tie-breaking in the sort step. We expected the proposed algorithm to create near ideal fold assignments, similar to the

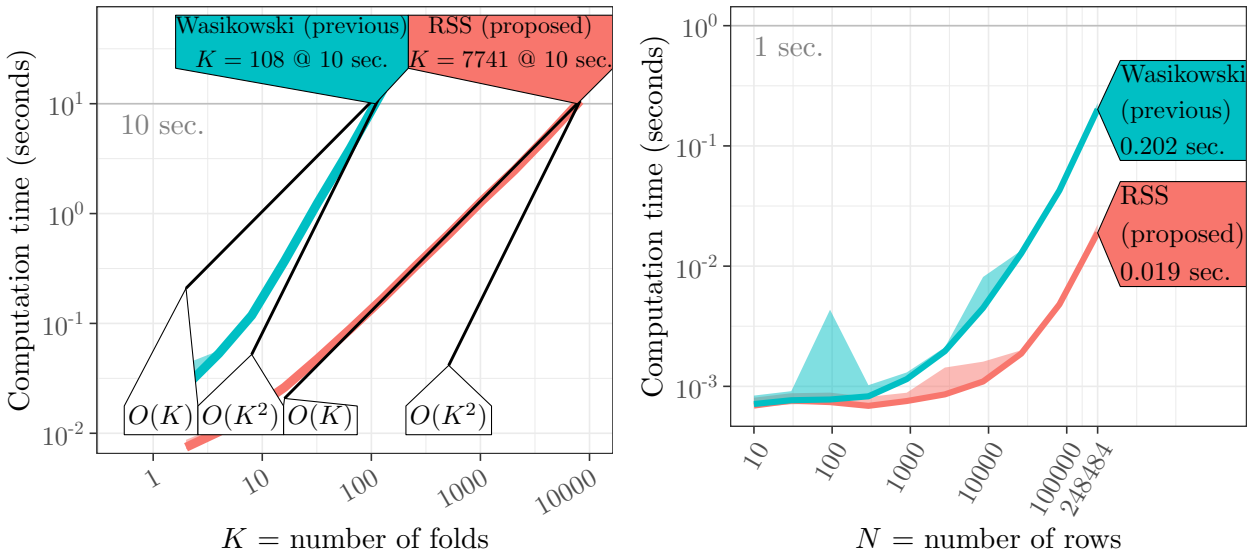


Figure 2: Empirical timings of the greedy fold assignment step of the previous and proposed algorithms, implemented in C++ code, run on Laribi2024 data (colored curves for mean, shaded band for min/max over 10 timings). **Left:** timings as number of folds K increases. Asymptotic reference curves (black), fit to largest empirical K values, indicate that previous Wasikowski is $O(K^2)$, and proposed RSS is $O(K)$, as expected. **Right:** timings as number of data N increases shows that RSS is ≈ 10 times faster asymptotically.

previous heuristic. For $K = 4$ folds and one random seed, Table 3 shows the counts obtained using four methods (ideal, proposed RSS, previous Wasikowski, random baseline). In agreement with our expectations, these results show that the proposed algorithm almost always yields fold assignments with nearly ideal counts, similar to the previous Wasikowski heuristic (both much better than random). The only exception was in AZtrees data, for which any algorithm with $K \geq 4$ folds can not get close to ideal, due to one large group with 1478 rows (however our algorithm works in these data with only $K = 3$ folds, Table 1 case E; and it works with $K = 2$ folds, Figure 1).

We also computed the root mean squared error between the ideal and actual counts per fold/stratum (Figure 1). In agreement with our expectations, we observed that proposed RSS and previous Wasikowski had comparably small error rates (sometimes one or the other slightly better, almost always much smaller error rates than random). The origami and rsample R packages require one stratum per group, so only run in the AZtrees data, but gave error rates similar to random (with data leakage for origami). The bioLeak package ran in all data, but almost always yielded error rates much larger than the proposed RSS and previous Wasikowski algorithms. Overall, these results indicate that the proposed RSS algorithm provides state-of-the-art fold assignment accuracy for a variety of real data sets with groups and strata.

5.4 Empirical timings

We expected the greedy fold assignment step of our proposed RSS algorithm to be asymptotically faster than the previous Wasikowski algorithm. We therefore ran empirical timings on the Laribi2024 data, varying number of folds K. We fit asymptotic references to the empirical timings (Figure 2, left), which indicated $O(K^2)$ time for the previous Wasikowski algorithm, and $O(K)$ time for the proposed RSS algorithm (as expected). We then varied number of rows N (keeping $K = 10$ folds constant), which gave a similar asymptotic slope on the log-log plot, but with the proposed RSS algorithm faster by a factor of about 10 (Figure 2, right). Overall, these empirical timings suggest that the proposed RSS algorithm is much faster than the previous Wasikowski algorithm.

6 CONCLUSIONS

In structured data sets with groups and strata, cross-validation fold assignment is NP-hard. We proposed a new greedy algorithm that has asymptotically less time and memory usage than the previous Wasikowski algorithm in scikit-learn. Our empirical results on synthetic and real data showed that our proposed algorithm is at least as accurate, and much faster, than the previous Wasikowski algorithm. Future work could consider other loss functions like a max difference [Pop and Matei, 2013], or adapting our algorithm for new applications such as imbalanced subset generation.

AI use statement

We have not used generative AI tools for any task related to this paper.

REFERENCES

S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statist. Surv.*, 4:40–79, 2010. ISSN 1935-7516. doi: 10.1214/09-SS054.

S. Asif, Y. Wenhui, S. ur Rehman, Q. ul ain, K. Amjad, Y. Yueyang, S. Jinhai, and M. Awais. Advancements and prospects of machine learning in medical diagnostics: unveiling the future of diagnostic precision. *Archives of Computational Methods in Engineering*, 32(2):853–883, 2025.

J. Coyle, N. Hejazi, I. Malenica, and R. Phillips. *origami: Generalized Framework for Cross-Validation*, 2026. R package version 1.0.8.

J. R. Coyle and N. S. Hejazi. origami: A generalized framework for cross-validation in r. *Journal of Open Source Software*, 3(21):512, 2018. doi: 10.21105/joss.00512.

H. Frick, F. Chow, M. Kuhn, M. Mahoney, J. Silge, and H. Wickham. *rsample: General Resampling Infrastructure*, 2026. R package version 1.3.2.

U. Halekoh, S. Højsgaard, and J. Yan. The r package geepack for generalized estimating equations. *Journal of Statistical Software*, 15/2:1–11, 2006.

B. Hayes. Computing science: The easiest hard problem. *American Scientist*, 90(2):113–117, 2002.

L. Hermida and R. Martynov. scikit-learn pull request 18649 on github, 2020.

T. D. Hocking, G. Thibault, C. S. Bodine, P. N. Arellano, A. F. Shenkin, and O. Lindly. SOAK: Same/Other/All K-fold cross-validation for estimating similarity of patterns in data subsets. *Statistical Analysis and Data Mining*, 19(1), 2026.

D. S. Johnson. Bin packing. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 207–211 in Encyclopedia of Algorithms. Springer New York, 2016. doi: 10.1007/978-1-4939-2864-4_49.

S. Kapoor and A. Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9), 2023.

S. Kaufman, S. Rosset, C. Perlich, and O. Stitelman. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4):1–21, 2012.

S. Korkmaz. *bioLeak: Leakage-Safe Modeling and Auditing for Genomic and Clinical Data*, 2026. R package version 0.3.8.

H. Laribi, N. Raymond, R. Taseen, D. Poenaru, and M. Vallières. Leveraging patients' longitudinal data to improve the hospital one-year mortality risk, 2024. Zenodo dataset 12954673.

H. Laribi, N. Raymond, R. Taseen, D. Poenaru, and M. Vallières. Leveraging patients' longitudinal data to improve the hospital one-year mortality risk. *Health Inf. Sci. Syst.*, 13(23), 2025.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. Ploton, F. Mortier, M. Réjou-Méchain, N. Barbier, N. Picard, V. Rossi, C. Dormann, G. Cornu, G. Viennois, N. Bayol, et al. Spatial validation reveals poor predictive performance of large-scale ecological mapping models. *Nature communications*, 11 (1):4540, 2020.

P. C. Pop and O. Matei. A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem. *Applied Mathematical Modelling*, 37(22):9191–9202, 2013. ISSN 0307-904X.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2026.

S. Raschka. Model evaluation, model selection, and algorithm selection in machine learning, 2020. URL `https://arxiv.org/abs/1811.12808`.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173, 2008.

sklearn developers. What's new in scikit-learn v1.0 (ChangeLog in GitHub repository), 2021.

J. Wasikowski. Stratified Group k-Fold Cross-Validation. https://www.kaggle.com/code/jakubwasikowski/stratified-group-k-fold-cross-validation/notebook, 2019.

J. Yang. Rethinking tokenization: Crafting better tokenizers for large language models, 2024. URL `https://arxiv.org/abs/2403.00417`.

CHECKLIST

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] section 3
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] sections 1,3, Table 2.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No] source code is currently public but not available during peer review, to respect double blind review.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable] no theorems in this paper.
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [No] source code is currently public but not available during peer review, to respect double blind review.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Not Applicable] no training involved, only split generation.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes] Figure 1 caption "10 random group orderings," Figure 2 caption " colored curves for mean, shaded band for min/max over 10 timings."
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] Section 5 All results were computed on a 2025 laptop (Intel Core Ultra 5 125U CPU.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [No] all data are freely available.
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable] no new assets.
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]