Instance selection by fast local set border selector

Norbert Jankowski[1] and Mateusz Skarupski[1]

Department of Informatics, Faculty of Physics, Astronomy and Informatics,
Nicolaus Copernicus University, Poland
email: norbert@umk.pl

Abstract. Prototype selection is one of the typical goals of machine learning, which aims to reduce the number of vectors in the training set. The local set border selector (LSBo) algorithm is presented as a Pareto optimal choice between the reduction power of the training set and the classification quality. Its complexity is $O(n^2)$, which means that it is not very advantageous for larger sets. This article presents the Fast LSBo algorithm, which is based on the original idea of the LSBo algorithm. After the applied conceptual changes, the algorithm has achieved a complexity of $O(m \log m)$. Additionally, the analysis of Fast LSBo on several data sets shows that its classification quality and reduction power remain statistically indistinguishable from the original LSBo algorithm.

1 Introduction

Let us assume that we have a learning data set $\mathcal{D} = \{\langle \mathbf{x}_i, y_i \rangle : i = 1, \ldots, m\}$ where $\mathbf{x}_i \in R^n$ are the input vectors and $y_i \in [1, \ldots, c]$ are the class labels. Selection of instances (prototypes) means that we are looking for a subset $S \subseteq \mathcal{D}$ that is enough to build a trustworthy classifier, for example, a k nearest neighbor (kNN) classifier [2]. Vector selection algorithms can be divided into two groups: filtering algorithms and prototype selection algorithms. The first group consists of algorithms whose main goal is to remove erroneous vectors, in other words, vectors that are inconsistent with the rest of the data set. Here, the best examples are the ENN [10] and LSSm algorithms [6]. The second group consists of algorithms that try to select prototypes, i.e. the most important vectors of the training set, i.e. those that carry the basic knowledge about the decision boundaries between classes. Interesting examples of such algorithms include [3, 9, 4]. It is also worth mentioning here a few review articles devoted to a broad analysis of vector selection algorithms [3, 4].

The first time *local sets* was introduced in the algorithm Iterative Case Filtering (ICF) in [1]. The *local set* for a given vector $\mathbf{x}$ from $\mathcal{D}$ is defined as the set of vectors in the largest hypersphere centered in $\mathbf{x}$ that does not contain an instance of the opposite class (an *enemy*): $LS(\mathbf{x}) = \{\mathbf{x}' : ||\mathbf{x}' - \mathbf{x}|| < ||\mathbf{x} - ne(\mathbf{x})||\}$, where $ne(\mathbf{x}) = \arg\min_{\mathbf{x}' \wedge y \neq y'} ||\mathbf{x} - \mathbf{x}'||$ is the nearest enemy. This definition strongly uses the distance to the nearest vector of an alien class as the radius of the hypersphere.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

Before presenting the local set border selector (LSBo) algorithm [6], it is necessary to present the local set-based smoother (LSSm) algorithm, which is the necessary first phase of the LSBo algorithm.

The LSSm algorithm is a very good example of a filtering algorithm, i.e., an algorithm that removes potentially inconsistent vectors from the original data set. Therefore, it will be used in the algorithm presented below, however in a modified version of lower complexity, but providing the same quality of selection in terms of its usefulness in classification.

The LSSm uses the idea of local sets to define two properties that are of vital importance for constructing the final algorithm. The first is the *usefulness* of a given instance $\mathbf{x}$, measured as the number of instances which has $\mathbf{x}$ in their local sets:

$$u(\mathbf{x}) = |\{\mathbf{x}' \in \mathcal{D} \ : \ \mathbf{x} \in LS(\mathbf{x}')\}| \tag{1}$$

The second property used in LSSm is the *harmfulness* of $\mathbf{x}$ measured as the number of instances for which the $\mathbf{x}$ instance is the nearest enemy:

$$h(\mathbf{x}) = |\{\mathbf{x}' \in \mathcal{D} \ : \ ne(\mathbf{x}') = \mathbf{x}\}| \tag{2}$$

The difference between those two properties defines the strength of balance between usefulness and harmfulness. The LSSm algorithm removes instances with greater harmfulness than usefulness. The final LSSm algorithm is presented in Alg. 1. The complexity of the LSSm algorithm is $O(m^2)$.

Now, the LSBo algorithm can be described. The first step of LSBo is to filter the original dataset $\mathcal{D}$ by the LSSm algorithm, thus obtaining a filtered dataset $\mathcal{T}$. The next step is to calculate local sets for $\mathcal{T}$. After that, the vectors in $\mathcal{T}$ are ordered (ascending) by the cardinality of local sets $|LS(\mathbf{x})|$. The initial set of prototypes S is empty and the main step browses the ordered vectors in the previous step, and if neither element of $LS(\mathbf{x})$ is already in S then S is extended by $\mathbf{x}$.

The algorithm's strategy first analyzes what is near to the decision boundaries after filtering out bad vectors. The prototype set is thus built from reliable vectors on the decision sides, and therefore, vectors far from the boundaries do not need to be added to the prototype set. The algorithm is presented in Alg. 2.

Algorithm 1: LSSm($\mathcal{D}$)

```
Data: D — dataset
Result: S
S = {}
compute local sets(D)
foreach x ∈ D do
    compute u(x) and h(x)
    if u(x) ≥ h(x) then
        S = S ∪ {x}
```

Algorithm 2: LSBo($\mathcal{D}$)

```
Data: D — dataset
Result: S
S = {}
T = LSSm(D)
compute local sets(T)
sort T ascending by the cardinality
  of LS
foreach x ∈ T do
    if x.LS ∩ S = {} then
        S = S ∪ {x}
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

2 Fast local set border selector algorithm

To construct the Fast LSBo algorithm, it was necessary to reduce the complexity of the LSSm algorithm and the complexity of the central part of the LSBo algorithm. A fast version of the LSSm algorithm was proposed by us in [5]. However, it will be slightly modified here.

The crucial point that the complexity of LSSm and LSBo is $O(m^2)$ is: $\sum_{\mathbf{x}\in\mathcal{D}} LSC(\mathbf{x}) = O(m^2)$. This means that as long as the algorithm uses the LS, the complexity will be $O(m^2)$. For this reason, one of the key changes for both algorithms is to use the firmly local sets (FLS) instead of local sets:

$$FLS(\mathbf{x}) = \begin{cases} \{\mathbf{x}' \ : \ ||\mathbf{x}' - \mathbf{x}|| < ||\mathbf{x} - ne(\mathbf{x})||\}, & \text{if } \exists_{\mathbf{x}'\in N^k(\mathbf{x})} \ y \neq y' \\ N^k(\mathbf{x}), & \text{otherwise,} \end{cases} \tag{3}$$

where $N^k(\mathbf{x})$ is a set of nearest neighbours of $\mathbf{x}$.

Fast LSSm algorithm: FLS is very crucial for overall complexity because since k is $O(1)$, then the $FLS(\mathbf{x})$ can be computed in average complexity $O(\log m)$. In FastLSSm, we used balanced forests of locality-sensitive hashing to compute nearest neighbours and LS's.

In this version of fast LSSm, the graph-based approximation (HNSW) of the nearest neighbours will be used [7]. This change is dictated by the requirements of the central part of the Fast LSBo algorithm presented in the next section.

The usefulness and harmfulness must be redefined as in [5] by

$$u'(\mathbf{x}) = |\{\mathbf{x}' \in \mathcal{D} \ : \ \mathbf{x} \in FLS(\mathbf{x}')\}|, \tag{4}$$

and by

$$h'(\mathbf{x}) = |\{\mathbf{x}' \in \mathcal{D} \ : \ ne(\mathbf{x}') = \mathbf{x} \ \wedge \ \mathbf{x} \in N^k(\mathbf{x}')\}| \tag{5}$$

appropriately.

The main idea behind the LSSm lies in: instance $\mathbf{x}$ remains in set if its usefulness is not smaller than harmfulness. See algorithm 3.

Hopefully, $|FLS(\mathbf{x})|$ has a more useful property: $\sum_{\mathbf{x}\in\mathcal{D}} |FLS(\mathbf{x})| = O(m)$ because k is fixed (not dependent on m). This proves vital to achieving an overall complexity of $O(m \log m)$, as discussed further in Section 3.

Fast LSBo algorithm: Now, all the elements of the fast LSBo algorithm will be presented. The LSBo algorithm is also based on the FLS definition instead of LS. However, several other modifications must be made to the original algorithm. The first is the use of the fast version of the LSSm algorithm – this is a necessary condition to have a chance of the overall complexity of $O(m \log m)$.

The next step is to use a unique structure to represent the set S, i.e., the set of selected prototypes. The representation of the set S will play an additional role here, apart from storing the selected vectors. The structure of the set S will be used to find the closest vectors—it will be necessary to know which vector in S is closest to a given $\mathbf{x}$.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

Algorithm 3: FastLSSm($\mathcal{D}$)

Data: $\mathcal{D}$ — dataset
Result: S
1 $S = \{\}$
2 construct HNSW set on $\mathcal{D}$
3 compute $N^k(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{D}$ using HNSW's
4 **foreach** $\mathbf{x} \in \mathcal{D}$ **do**
5 find nearest enemy $ne(\mathbf{x})$ in $N^k(\mathbf{x})$
6 **if** *any enemy* **then**
7 $h'[ne(\mathbf{x})]$++
8 **foreach** $\mathbf{x} \in \mathcal{D}$ **do**
9 **foreach** $\mathbf{x}' \in N^k(\mathbf{x})$ **do**
10 **if** *no enemy of* $\mathbf{x}$ *in* $N^k(\mathbf{x})$ $\vee$ $||\mathbf{x}' - \mathbf{x}|| < ||\mathbf{x} - ne(\mathbf{x})||$ **then**
11 $u'[\mathbf{x}']$++
12 **foreach** $\mathbf{x} \in \mathcal{D}$ **do**
13 **if** $u'(\mathbf{x}) \geq h'(\mathbf{x})$ **then**
14 $S = S \cup \{\mathbf{x}\}$

That is why the graph-based HNSW structure [7] represents S, which allows alternating between adding new elements to the set and querying for nearest neighbors. As we show below, this plays a key role in the most essential point of the LSBo algorithm.

The second essential data structure is the HNSW (J) array of sets. This structure is used twofold: to determine nearest enemy class vectors and to determine the cardinality of the FLS sets. Determining the nearest enemies consists of determining each set's nearest neighbor from the J array of sets except for the own set. On the other hand, building the FLS sets consists of taking the nearest k neighbors from the own class set from the J array, but only those closer than the nearest enemy.

The fast version of the LSBo algorithm also uses the ordering of vectors in T in the main loop, which determines the prototype vectors S. This time, the ascending sorting occurs here by the cardinality of FLS instead of the cardinality of LS. In the above-specified order, all vectors from $\mathcal{T}$ are then traversed. However, the condition from the line 6 of the LSBo algorithm 2 had to be significantly reformulated. This condition in LSBo stated whether no vector from $LS(\mathbf{x})$ is in the prototype set of S because if so, $\mathbf{x}$ is necessary for S. However, in the fast version of LSBo, we do not have the set $LS(\mathbf{x})$. Still, the same effect can be obtained by the condition:

$$||\mathbf{x} - \mathbf{x}_n|| > ||\mathbf{x} - \mathbf{x}_e||,$$

in which it is tested whether the nearest neighbor of $\mathbf{x}$ in S ($\mathbf{x}_n$) is more distant than the nearest enemy of $\mathbf{x}$ ($\mathbf{x}_e$). Such a case means no vector from the *real* $LS(\mathbf{x})$ is in the current set S. Consequently, it forces the addition of the vector $\mathbf{x}$ to S.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

Algorithm 4: FastLSBo($\mathcal{D}$)

Data: $\mathcal{D}$ — dataset
Result: S — the set of selected prototypes
1 $\mathcal{T} = FastLSSm(\mathcal{D})$
2 S = empty HNSW set
3 J = construct one HNSW per class in $\mathcal{T}$ (one HNSW set for all instances of given class)
4 **foreach** $\mathbf{x} \in \mathcal{T}$ **do**
5 compute $|FLS(\mathbf{x})|$ using HNSW for class y from J
6 sort $\mathcal{T}$ ascending by $|FLS(\mathbf{x})|$
7 **foreach** $\langle \mathbf{x}, y \rangle \in \mathcal{T}$ **do**
8 $\mathbf{x}_n$ = nearest neighbour in S to $\mathbf{x}$
9 $\mathbf{x}_e$ = nearest enemy of $\mathbf{x}$ in J
10 **if** $||\mathbf{x} - \mathbf{x}_n|| > ||\mathbf{x} - \mathbf{x}_e||$ **then**
11 $S = S \cup \{\mathbf{x}\}$

3 Complexity and experimental results on benchmark

This section starts with the presentation of the average complexity of Fast LSBo is $O(m \log m)$ and that the accuracy of the fast version of LSBo is so strongly similar (in almost all cases the same) to the original LSBo. The most expensive part of the Fast LSSm algorithm is its first part, i.e. the lines 2 and 3. The average cost of adding m vectors to the HNSW structure is $O(m \log m)$. The cost of determining k nearest neighbors for all m vectors is also $O(m \log m)$. In figure 1, a relation is presented between the time of adding vectors and building sets of nearest neighbors using HNSW. On the OX axis we see the values of the number of m vectors in the set $\mathcal{D}$. On the OY axis is the execution time divided by $m \log m$. This means that the presented graph should not grow if building and using the HNSW architecture is to have complexity $O(m \log m)$ as mentioned above. And this is precisely how this graph behaves, which proves the complexity of $O(m \log m)$.

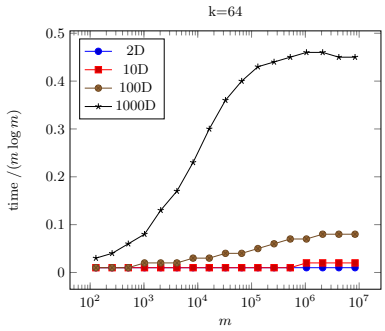
The next three loops of the algorithm have a complexity of $O(m)$ (we assume that k is a constant value).

The Fast LSBo algorithm starts execution with the LSSm algorithm.

In the next part, an empty set S is created, to which a subset of $\mathcal{D}$ will be added. Consequently, this will not worsen the previous complexity. In the next part, a series of HNSW structures (table J) is created, in which the vectors of the appropriate classes will be added (one HNSW structure $J[i]$ for the vectors of i-th class). Then, the sets $FLS(\mathbf{x})$ are determined. The costs of these operations are not greater than the costs of creating an HNSW structure for the entire set $\mathcal{D}$. This means that the complexity of this part is limited by $O(m \log m)$ as well as sooner. The next step is to sort the array of all cardinalities $|FLS(\mathbf{x})|$—once again $O(m \log m)$.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

Fig. 1. Time consumption by HNSW.



In the last loop, we have m iterations, and each iteration has complexity $O(\log m)$ because none of the loop elements require more complexity than $O(\log m)$.

This gives the complexity $O(m \log m)$ of the fast LSBo algorithm.

Experimental results: Since it was shown in [6] that the LSBo algorithm is very effective as a Pareto tradeoff between training set reduction power and classification quality, these tests compare LSBo with state-of-the-art instance selection algorithm and kNN. Hence, here we focus on showing that the classification quality and the quality of prototype selection remain almost the same.

For this purpose, 45 sets were selected from the UCI Machine Learning Repository [8]. Datasets differ in the number of instances, attributes, and balance of classes. In all tests, we used 10-fold stratified cross-validation and all learning machines were trained on the same sets of data partitions. To visualize the performance of all algorithms, we present the average accuracy for each benchmark dataset and each learning machine, Table 1. In addition, we present the average reduction of the dataset size in separate tables. *Ranks* are calculated for each machine for a given dataset. The ranks are calculated as follows: First, for a given benchmark dataset, the averaged accuracies of all learning machines are sorted in descending order. The machine with the highest average accuracy is ranked 1. Then, the following machines in the accuracy order whose accuracies are not statistically different[1] from the result of the first machine are ranked 1, until a machine with a statistics different result is encountered. That machine starts the next rank group (2, 3, and so on), and an analogous process is repeated on the remaining (yet unranked) machines. Notice that each cell in the main part of is in a form: $acc + std(rank)$, where acc is the average accuracy (for a given data set and given learning machine), std is its standard deviation and $rank$ is the rank described just above. If a given table cell is in bold, it means that this result is the best for the given data set or not worse than the best one (rank 1 = winners).

[1] We use the paired t-test to test the significance of statistical differences.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

As can be seen from the results presented in the table, the differences between the classification quality of the regular version of the LSBo algorithm and the fast version are practically negligible for all data sets. The same is true for the reduction power of the training set—both algorithms work very similarly in this aspect. The goal of creating a fast version of the LSBo algorithm has been achieved.

Conclusions

The research goal was to create an algorithm that would have lower complexity than the local set border selector algorithm and that would be as good as possible in terms of classification quality on benchmark tests. Thanks to the careful selection of new data structures and appropriate reformulation of the original algorithm, it was possible to create a new algorithm that is just as good but with complexity $O(m \log m)$ instead of $O(m^2)$. Thanks to this, it will be possible to use this algorithm for much more significant problems.

References

1. Brighton, H., Mellish, C.: Advances in instance selection for instance-based learning algorithms. Data Mining and Knowledge Discovery **6**(2), 153–172 (2002)
2. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. Institute of Electrical and Electronics Engineers Transactions on Information Theory **13**(1), 21–27 (Jan 1967)
3. Garcia, S., Derrac, J., Cano, J., Herrera, F.: Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(3), 417–435 (2012)
4. Jankowski, N., Grochowski, M.: Comparison of instances selection algorithms: Ii. Algorithms survey. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science, vol. 3070, pp. 598–603. Springer-Verlag, Poland, Zakopane (2004), http://www.is.umk.pl/ norbert/publications/04-zakopane-NJMG.pdf
5. Jankowski, N.: A fast and efficient algorithm for filtering the training dataset. In: Neural Information Processing. vol. 13623, pp. 504–512. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-30105-6˙42
6. Leyva, E., González, A., Pérez, R.: Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. Pattern Recognition **48**(4), 1523–1537 (2015). https://doi.org/10.1016/j.patcog.2014.10.001
7. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. **42**(4), 824–836 (2020). https://doi.org/10.1109/TPAMI.2018.2889473
8. Merz, C.J., Murphy, P.M.: UCI repository of machine learning databases (1998), http://www.ics.uci.edu/∼mlearn/MLRepository.html
9. Wilson, D.R., Martinez, T.R.: Reduction techniques for instance-based learning algorithms. Machine Learning **38**(3), 257–286 (2000)
10. Wilson, D.: Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics **2**(3), 408–421 (1972)

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17

Table 1. Comparison of LSBo and fast LSBo algorithms.

Dataset	Accuracy		Removed %	
	LSBo	LSBoFast	LSBo	LSBoFast
arrhythmia	**44.2±20(1)**	**43.6±19(1)**	0.55±0.04	0.55±0.04
autos	**60.3±12(1)**	**59.2±12(1)**	0.67±0.02	0.67±0.02
balance-scale	**75.9±5.7(1)**	**75.3±5.3(1)**	0.78±0.01	0.84±0.01
blood-transfusion	**70±5.1(1)**	**68.7±5.8(1)**	0.75±0.01	0.86±0.01
breast-cancer-diagnostic	**92.8±4(1)**	**92.5±3.8(1)**	0.91±0.004	0.91±0.005
breast-cancer-original	**94.5±4.5(1)**	**94.6±3.6(1)**	0.97±0.004	0.96±0.004
breast-cancer-prognostic	**63.3±13(1)**	**63.4±13(1)**	0.78±0.03	0.78±0.03
breast-tissue	**58.7±14(1)**	**59.5±14(1)**	0.73±0.02	0.73±0.02
car-evaluation	**87.4±2.2(1)**	78.4±2.3(2)	0.52±0.01	0.89±0.005
cardiotocography-1	**68.6±3.5(1)**	**68.5±3.7(1)**	0.71±0.004	0.71±0.004
cardiotocography-2	**85.9±2.1(1)**	**86±2.3(1)**	0.9±0.003	0.89±0.004
chess-rook-vs-pawn	**85.4±1.8(1)**	**85.5±2(1)**	0.8±0.004	0.79±0.004
cmc	**42.6±3.8(1)**	**42.8±3.7(1)**	0.61±0.007	0.58±0.008
congressional-voting	**89.1±6.5(1)**	**89.4±6.2(1)**	0.88±0.01	0.88±0.01
connectionist-bench-sonar	**78.3±9.2(1)**	**78.2±9.3(1)**	0.7±0.01	0.71±0.01
connectionist-bench-vowel	**72.1±5.8(1)**	**72.2±5.7(1)**	0.78±0.005	0.78±0.005
cylinder-bands	**65.9±8.8(1)**	**66.3±8.7(1)**	0.69±0.01	0.7±0.01
dermatology	**87.7±5(1)**	**87.9±5.4(1)**	0.8±0.009	0.81±0.009
ecoli	**77.1±6.9(1)**	**77.5±7.2(1)**	0.85±0.01	0.85±0.01
glass	**62.5±9.4(1)**	**62.3±9.2(1)**	0.72±0.01	0.73±0.01
habermans-survival	**66.6±7.8(1)**	**67.1±8.1(1)**	0.77±0.01	0.78±0.01
hepatitis	**80.9±11(1)**	**81.1±12(1)**	0.84±0.03	0.84±0.03
ionosphere	**82±6.4(1)**	**81.8±6.5(1)**	0.82±0.01	0.81±0.01
iris	**89.7±8.2(1)**	**88.4±9.2(1)**	0.88±0.009	0.89±0.01
libras-movement	**68.7±6.7(1)**	**68.5±7.4(1)**	0.68±0.01	0.69±0.01
liver-disorders	**60.7±9(1)**	59.5±9.3(2)	0.61±0.02	0.62±0.01
lymph	**74.4±12(1)**	**73.4±11(1)**	0.74±0.02	0.79±0.02
monks-problems-1	91.8±7(2)	**93.3±6.1(1)**	0.8±0.001	0.8±0.002
monks-problems-2	**54.7±6.4(1)**	**55.3±5.4(1)**	0.64±0.01	0.64±0.02
monks-problems-3	**90.6±4.7(1)**	**91±3.9(1)**	0.8±0.003	0.81±0.004
parkinsons	**84.7±8.6(1)**	**84.1±8.9(1)**	0.84±0.01	0.84±0.01
pima-indians-diabetes	**69.1±5(1)**	**68.9±5.3(1)**	0.75±0.008	0.75±0.008
sonar	**78.3±9.2(1)**	**78.2±9.3(1)**	0.7±0.01	0.71±0.01
spambase	**85.4±1.9(1)**	**85.4±1.9(1)**	0.88±0.003	0.87±0.005
spect-heart	**74±9(1)**	**72.2±9.4(1)**	0.79±0.01	0.91±0.01
spectf-heart	69.6±9(2)	**70.7±8.5(1)**	0.79±0.02	0.79±0.02
statlog-australian-credit	**72.8±6(1)**	**73.4±5.7(1)**	0.77±0.01	0.78±0.01
statlog-german-credit	**67±4.6(1)**	**67±4.4(1)**	0.7±0.008	0.71±0.008
statlog-heart	**76.1±7.4(1)**	**75.3±8(1)**	0.75±0.01	0.75±0.01
statlog-vehicle	**67.2±4.8(1)**	**66.8±4.8(1)**	0.7±0.008	0.71±0.007
teaching-assistant	**48.3±14(1)**	**48.9±13(1)**	0.59±0.02	0.59±0.02
thyroid-disease	91±1.4(2)	**91.6±1(1)**	0.97±0.001	0.96±0.003
vote	**91.2±5.6(1)**	89.6±6.5(2)	0.86±0.02	0.92±0.01
wine	**92.5±6.2(1)**	**92.4±6.5(1)**	0.83±0.01	0.84±0.01
zoo	**78.1±10(1)**	**78.1±11(1)**	0.89±0.006	0.89±0.006
Mean	74.8±7.2	74.5±7.2	0.77±0.01	0.78±0.01
Mean Rank	1.07±0.038	1.07±0.038		

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97635-3_17