Cost-sensitive Hierarchical Clustering for Dynamic Classifier Selection

Meinolf Sellmann  
InsideOpt  
New York, USA  
meinolf@gmail.com

Tapan Shah  
Machine Learning  
GE Global Research  
San Ramon, USA  
tapan.shah@ge.com

Abstract—Given an ensemble of classifiers, dynamic classifier selection (DCS) selects one classifier depending on the particular input vector that we get to classify. DCS is a special case of algorithm selection (AS) where we can choose from multiple different algorithms to process a given input. We investigate if cost-sensitive hierarchical clustering (CSHC), a method originally developed for AS, is suited for DCS. We tailor CSHC for the special case of choosing a classification algorithm and compare with state-of-the-art DCS methods. We then show how the new methodology can be used for stacking. Experimental results show that CSHC-based DCS outperforms the best methods to date.

Index Terms—ensembles, dynamic classifier selection, portfolios

I. INTRODUCTION

When multiple classifiers are available to us, which may be based on different concept classes or may themselves be ensembles, we may want to choose dynamically, after seeing the feature input, which classifier to use. A method is needed to choose a classifier. This problem is known in the literature as dynamic classifier selection (DCS).

There are other ways to combine multiple classifiers, for example by using each classifier’s support for each of the possible class labels and aggregating this information. This is the basic idea behind stacking [1]. Note that, in stacking, the final class label may not coincide with any of the classes chosen by any of the base classifiers, which gives this technique more flexibility and the potential to outperform dynamic classifier selection. However, there are two main disadvantages of this more flexible aggregation method: 1. It requires running all classifiers in the ensemble, thereby increasing the energy consumption. 2. Explanations for the result of an aggregation cannot be inherited from the base classifiers as the result may be a class that no base classifier chose.

Consider two examples: 1. Consider a classifier that detects credit card fraud using a stacked classifier. A company like Mastercard handles 420 million transactions per day. Reducing the cost of running an ensemble used for fraud detection by a mere 20ms per call to the model (which is conservatively what the method described in this paper achieves) would prevent over 106 tons of CO2 annually [2] - for this one model at this one company.

2. Consider a medical imaging application, e.g. for skin cancer detection. [3] show that ensembling image classifiers significantly improves performance. Each image classifier may provide a part of the image to justify the classification. When using DCS, we can simply inherit this justification from the image classifier selected (potentially augmented by an explanation why we chose this classifier). Using stacking, however, the final classification may not coincide with the classification of any base classifier, leaving us with no part of the image we can present to the diagnostician to support the final recommendation.

DCS addresses both disadvantages of ensembling and can be viewed as a special case of algorithm selection (AS), the core operation in algorithm portfolios [4]. One method for selecting an algorithm out of a portfolio of algorithms was introduced in [5] and employs cost-sensitive multi-classification for algorithm selection. In this paper, we investigate whether this AS approach can be used effectively for DCS. We introduce several modifications to make the method more suited for classifier selection. Then, we compare the approach with state-of-the-art DCS methods. Finally, we also show how the method can be used to realize dynamic stacking.

II. DYNAMIC CLASSIFIER SELECTION

State-of-the-art DCS methods work by estimating the competence of the base classifiers for a given query sample and then select the base classifier with highest competence. The competence is commonly estimated as follows:

1) For a given sample, a region of competence i.e. a local neighborhood of training samples is computed using either k-nearest neighbors (k-NN) or clustering methods.
2) Then, the competence level of each classifier is computed on the neighborhood, based on varying criteria like accuracy of base classifiers, ranking, etc.

Examples that realize the framework above, and are used as baselines, are Local Class Accuracy (LCA) [6], Overall Local Accuracy (OLA) [6], A Priori [7], A Posteriori [7], and Multiple Classifier Behavior (MCB) [8]: We also compare with Majority Voting (MV) where all the base classifiers are pooled and the final class is the one selected by most base classifiers. Finally, we compare with stacking [1]: Probabilities assigned to all classes by base classifiers are combined by another (static) classifier to obtain the final prediction. We use Stack-LR and Stack-DT which use as second-stage classifier logistic regression and decision tree [9], respectively. For
further details, we refer to the very thorough discussion of various dynamic classifier and ensemble selection methods in [10].

III. COST-SENSITIVE HIERARCHICAL CLUSTERING

The idea behind CSHC [5] is to recursively split a cluster of input samples such that the inputs within a partition can agree on one algorithm that shall be used to process all inputs in the respective partition. CSHC therefore builds a decision tree. However, it does not use entropy to determine splitting features and values, but considers the overall performance when using a different, optimal algorithm for each partition, rather than the same algorithm on all examples in the parent cluster. The split that results in the best performance gain is then selected.

Note that performance can be any metric desired, from running time (typically the target in search and optimization), to optimality gap within a fixed time frame (e.g., when tuning local search heuristics), to some other metric of quality. For the purpose of classifier selection, we will simply use the method’s accuracy. Three hyper-parameters guide when the recursive splitting of clusters stops. The first is a simple depth limit, the second a minimum number of samples that must remain in each cluster, and the last is a minimum improvement that is expected from splitting a cluster.

As with decision trees, it has been found beneficial to build more than one hierarchical clustering. Identical to random forests, in CSHC, for each new clustering, only a subset of features are allowed to be used to split the inputs, and a sub-sample (with replacement) is built from the total set of inputs to be clustered. Three hyper-parameters guide this process of ensembling clusterings: How many clusterings (trees) to construct, how many features are randomly selected to be used for splitting the sample set, and how often we sample the training inputs with replacement.

CSHC selects the algorithm that has the best cumulative rank over all clusterings. That is, when we are given a new input at test time, we determine which clusters the input falls into for each of the clusterings. Then, we rank all algorithms for each cluster and select the algorithm that has the best cumulative rank. For further details on CSHC, please see [5].

Note how CSHC differs from existing DCS methods. First, the multiple hierarchical clusterings built by sub-sampling the training samples with repetition creates neighborhoods (the multi-set of examples in clusters the target feature vector is assigned to) that give different weights to different training examples by including samples as many times as they appear in target clusters. Second, the clusterings are constructed not by considering unsupervised metric regions in the feature-space, or regions where the original machine learning problem favors the same class, but by considering regions which are handled well by the same classifier. And third, the performance is assessed by ranking classifiers on multiple clusters and picking the best, which is unlike how any other existing method determines the final selection.

IV. CSHC FOR DYNAMIC CLASSIFIER SELECTION

CSHC can be applied to any AS problem and is directly applicable to DCS as well. However, certain aspects make classifier selection a special case of general algorithm selection. In this section, we discuss these differences and propose some modifications to the vanilla CSHC methodology.

A. Training Data

The first particularity of DCS is the way how the training data is generated. When building an algorithm selector for SAT, for example, we run the various algorithms on each training instance and thereby gather the cost data needed to train the clusterings with CSHC. That is to say that, in other applications, the training instances used to train the selector usually have no influence on the algorithms in the portfolio.

This is not the case when using an algorithm selector for classifier selection. There is a certain amount of labeled data available, which needs to be used for training the base classifiers as well as the classifier selector. Obviously, the selector could be over-confident with a classifier if it only had access to cases where the classifier labels samples that were used to train the respective classifier. To circumvent this issue, we conduct a three-fold cross validation. In each fold, we use two thirds of the training data to train a classifier, then we evaluate the classifier on the remaining third of the data. The cost labels generated for CSHC are then exclusively derived from the validation performances. Note that, in this way, we can use the entire training data for generating clusterings.

B. CSHC-Rank Regression

Another difference is that, in general algorithm selection, we usually cannot run all algorithms. Imagine a case where we need to choose the fastest scheduler for a given scheduling instance. Running all schedulers is obviously not an option, we have to choose one before we see the algorithm output.

In the context of ensemble learning, the situation may be different. Of course, there may be scenarios where running all classifiers is too costly, for example because of latency requirements or because it is simply too cost-prohibitive to run them all. In this case, we can simply use the cumulative ranking procedure from CSHS. We will report on the performance of this method in the experimental results.

In other cases, however, our prime concern is classification performance rather than computational cost. Then, we may want to run all classifiers and use the classification results as well as the original features to select a classifier (and the associated class this classifier labels the input with). In the following, we present methods how to use this information in the context of CSHC.

One way how we can use the labels produced by the different classifiers is by voting. To this end, each classifier is assigned a certain weight, and the class it labels the input with gets this weight added as support. We select one of the classifiers that labels the input with the class that has the most support. Among all classifiers that lend the support, we select the one with the largest weight (with ties broken randomly).
We utilize the ranks that CSHC provides to assign a weight to each classifier. Particularly, we assign the cumulative rank over all clusterings (with the better classifiers having higher rank) as the weight for each classifier. Note that the clusters are input-specific. Therefore, the weight each classifier is assigned changes dynamically from test sample to test sample.

\section{C. Linear Programming-based Weighting}

We can also compute a set of weights that optimizes the performance over the multi-set of samples over all clusters the given feature vector is assigned to by CSHC. We propose to set up a linear program (LP) for this purpose.

Assume we are given the number \( n \in \mathbb{N} \) of different classifiers in the ensemble, the number \( C \in \mathbb{N} \) of classes, the set \( E = \{e_1, \ldots, e_k\} \) of unique samples in the union of all clusters the given feature vector falls into, the correct labels \( y_i \in \{1, \ldots, C\} \), as well as numbers \( m_i \in \mathbb{N} \) for \( 1 \leq i \leq k \) that determine how often example \( e_i \in E \) appears in the multi-set of samples returned by CSHC of the given feature vector. Finally, assume that, for each classifier \( a \in \{1, \ldots, n\} \) and each sample \( e_i \in E \), we are given the label \( l_{ia} \in \{1, \ldots, C\} \).

The LP we set up has three sets of variables. First, for each classifier \( a \in \{1, \ldots, n\} \), a weight \( 0 \leq w_a \leq 100 \) with \( a \in \{1, \ldots, n\} \). Moreover, for each unique example \( e_i \in E \), we introduce two penalty variables \( g_i, f_i \geq 0 \). We impose the following constraints: First, the weight variables must sum to 100: \( \sum_{c \leq C} w_c = 100 \). Next, for each unique example \( e_i \in E \) and each class \( c \in \{1, \ldots, C\} \) with \( c \neq y_i \), we add two constraints: \( g_i + \sum_{a \in C} w_a - \sum_{b \neq c} w_b \geq \gamma \), for some user-defined parameter \( 0 \leq \gamma < 100 \), and \( f_i + \sum_{a \in C} w_a - \sum_{b \neq c} w_b \geq 1 \). Then, we solve the LP to obtain weights and penalties that minimize the total penalty \( \sum_{i \leq k} m_i (g_i + 2f_i) \).

The LP favors classifier weightings that result in a support for the correct label that is at least \( \gamma \% \) more than the maximal support for any other label. When that is not possible, the LP strives to have at least the largest support for the correct label, or to get as close to the largest support as possible. For each example for which the weighted aggregate results in a class label that is correct, there is no extra penalty. Otherwise, the penalty is two times the gap of the total support for the wrong label minus the support for the correct label, plus whatever is needed to bring the gap between the support for the correct class to any other class to at least \( \gamma \).

As previously, based on the weights obtained, we compute the class that has the most aggregate support. We ultimately select the classifier that has the maximum weight among all that label the input with that maximally supported class. Note that such a classifier is guaranteed to exist.

\section{D. Confidence Assessment and Recourse}

A robust selection mechanism, inspired by [11], may be obtained by employing a process that considers how confident each of the classifier selection methods introduced above actually is (see Alg. 1). To this end, we consider the ratio between the class that has the second largest support and the class that has the largest support (based on the respective ways to compute the weights of each classifier, either by cumulative rank or by solving the linear program). The lower this ratio (we refer to this parameter with \( \rho \)), the higher our confidence that this selection is correct. We propose to use rank regression first, since it is computationally much cheaper than solving a linear program for each test sample. If the confidence in the rank selection is sufficient, we return the classifier selected. If the confidence is insufficient, we next compute the classifier selected by the LP weighting scheme. We assess the confidence in this method as well. If confidence is high enough, we return the respective classifier.

If confidence is also low in the LP-based weighting method, then we proceed as follows: If the class label of the classifier chosen by both rank regression and LP-based weighting are the same, then we return the classifier whose respective selection method has higher confidence (note that the class labels may be the same even when the two methods choose a different classifier). If the classes are not the same, we next check if the classifier returned by the original CSHC labels the input with the same class as one of the other two classifiers. If so, we return that classifier. If this also fails, which implies that all three selection methods select a different classifier and all three classifiers label the input with a different class, then we finally compute the dominant class label in the multi-set of training samples returned by CSHC. If one of the three classifiers provided by the three methods labels the input with that dominant class, then we choose this classifier. Otherwise, we return the classifier chosen by the LP-based weighting scheme.

\section{E. Dynamic Stacking}

As our final algorithmic contribution, we show how the LP-based weighting scheme can be used to realize a dynamic
stacking approach. We again use ranked-based classifier selection and return the corresponding class if confidence is high. When confidence is insufficient, we again employ LP-based weighting, yet this time using the probabilities $p_{aj}$ in $[0,1]$ that classifier $a$ assigns to class $j$. Using the same variables, objective, and notation as before, we impose the following constraints: $\sum_{c \leq C} w_c = 100$, and for each unique example $e_i \in E$ and each class $c \in \{1, \ldots, C\}$ with $c \neq y_i$, we add two constraints: $x_i + \sum_{a \neq a_i} p_{a,y_i} w_a = \sum_{b \neq y_i} p_{b,c} w_b \geq \gamma$ and $f_i + \sum_{a \neq a_i} p_{a,y_i} w_a - \sum_{b \neq c} p_{b,c} w_b \geq 1$. The only change is therefore that the support of each class is no longer limited to aggregating the weights of all classifiers that prefer the respective class over all others, but the weight of each classifier is spread over all classes in accordance to the probabilities the respective classifier assigns to each class.

V. NUMERICAL RESULTS

We now describe the experiments we conducted to quantify and compare performances.

Data Sets: We use 40 data sets from OpenML [12], [13] for our experiments. Due to space constraints, detailed information on each benchmark dataset is skipped.

Base Models: For each benchmark, we built a set of 5 base classifiers, Naïve Bayes (NB), Support Vector Classification (SVC), Perceptron, k-Nearest Neighbors (k-NN), and Decision Tree Classifier (DTC).

Competitors: We use the methods reviewed in Section II, with the exception of A Posteriori (APO) and KNORA-E (KE) which we found were significantly outperformed by their respective sister methods, APR and KU. Instead, we include a simple majority vote (MV) on the neighborhood instances. We use Python library DESLIB [14].

We compare with three variants based on CSHC. The first, referred to as CSHC, uses the unmodified AS method, trained on data obtained as described in Section IV-A. The second, referred to as CSHC-LPR, uses all modifications as introduced in Sections IV-A-IV-D. The third, referred to as Stack-CSHC, uses modifications Section IV-A-IV-E. Note that CSHC and CSHC-LPR are dynamic classifier selectors, while Stack-CSHC is a dynamic stacking method.

CSHC has six hyper parameters. We generate 50 trees using the number generator from [15]. For each tree, we sample with repetition from the training set until we obtain a multi-set of samples which amounts to 80% of the total training set of unique samples. To create each tree, we use two times the square root of all features, chosen uniformly at random. The last three hyper parameters determine when we stop the hierarchical refinement of clusters. First, we enforce that at least 2 samples remain within each cluster. Second, we limit the depth of the trees to be at most 15. And finally, we stop refining the clusters when the improvement by an additional split drops below 2%. For the LP-based weighting scheme, we set $\gamma = 80$. The recourse threshold $\rho$ is set to 0.5, which means that we only trust a classifier selection method outright when the support for the highest ranked class is at least twice that of the second most supported class. Note that all these parameters are set to the same values for all benchmarks we consider in the experiments. Naturally, these hyper-parameters could be tuned for each benchmark individually, for example by means of a cross validation. To demonstrate the effectiveness of the method proposed, we leave all CSHC parameters and the parameters for the modifications we introduced at the same default values for all benchmarks. For all other selection methods, we use the DESLIB library defaults for all hyper-parameters [14].

A. Highest Rank vs. Rank Regression

We begin our experimentation by comparing vanilla CSHC with the rank regression scheme we introduced in Section IV-B. Recall that CSHC selects the classifier that has the highest average rank over all clusters the test sample falls into. The rank regression modification we introduced, on the other hand, uses these average ranks as weights for the support each classifier gives to their favorite class.

Using our five base classifiers, we build ensembles using vanilla CSHC and with our newly introduced rank regression. Out of the 40 head-to-head comparisons, the rank regression wins 32 and loses 8. This confirms our initial speculation that using the actual classifications of each classifier to select the top classifier gives an advantage. However, note that this additional performance comes at the cost of having to run all classifiers first. CSHC, on the other hand, selects one classifier based on the original features, and thus only requires one base classifier to run.

B. Using Recourse at Low Confidence

In Table I, we show the performance of the confidence-assessment-and-recourse process we introduced (CSHC-LPR). The last rows give the number of wins/losses over CSHC-LPR, the MGL average ranks over all methods and benchmarks (the higher the rank, the higher the relative performance), and average accuracies over all 40 datasets. Competing method acronyms are defined in Section II. Recall from the earlier description that we run the rank regression first and, provided confidence is high, we move on directly with the classifier selected. Only when confidence is low, we employ the LP-based weighting method. If its confidence is high, we use the selected classifier, otherwise we consider the classifier with the highest average rank and eventually the majority class in the neighborhood to break the tie.

The last four rows in the table compare every other method with this one. In the fourth row from the bottom, we highlight the average rank over all benchmarks and competing approaches. In the second to last row, we show

$$MGI(X) \leftarrow \frac{\text{GM}(\text{Accuracy}_{\text{CSHC-LPR}})}{\text{Accuracy}_X} - 1,$$

in percent, where $X$ is the method compared with, and $\text{GM}(\cdot)$ is the geometric mean of a vector. Note that the $MGI(X)$ is the same as the ratio of the geometric mean of the accuracy of CSHC-LPR over the geometric mean of the
accuracy of method X, minus 1. Therefore, a value greater zero indicates that method X has lower accuracy than CSHC-LPR on average.

We observe that using confidence assessments helps make the selection more robust. CSHC-LPR significantly outperforms CSHC, both in terms of wins/losses over the 40 benchmarks, as well as in terms of the accuracy comparison as measured by MGI.

Diving deeper into our logs, we find that recourse is invoked very rarely. On bank-marketing, e.g., only for 345, or 3.8%, test samples, confidence in the initial rank regression is too low and the LP-based weighting method is invoked. This means that, given the default threshold $\rho = 0.5$, in over 96% of the test cases the support of the top class is at least twice as high as the class with second largest support, which in turn implies that the support for the top class is at least 66%. This illustrates how effective CSHC is at partitioning the feature space in such a way that the choice of classifier is clear.

At the same time, since most test samples can be handled by rank regression with high confidence, the method works very efficiently. The largest benchmark is bank-marketing with 16 features and close to 18,000 train and 9,000 test samples per fold (the fold actually has 36,000 training samples, but recall that we only allow 50% of training examples to be used to stay consistent with the way DESLIB trains the competitors). Generating the 50 clusterings takes around 11 sequential CPU seconds (KU 5s) which is manageable, especially since the training of different trees can be parallelized easily with linear speed-up. On hill-valley, which has 100 features and around 400 train and test samples, CSHC takes around 0.45 seconds to train. (KU 0.05s). At the same time, CSHC-LPR requires only around 1 millisecond to classify each sample (compared to KU with needs 106ms). Over all 40 benchmarks, CSHC-LPR needs 36ms per test sample, while KU needs 56ms.

When energy consumption is an important factor, for example when running mass-applications, then these savings are very meaningful in terms of $\text{CO}_2$ that is saved.

C. Comparison With the State of the Art

In Table I, we compare the original CSHC and CSHC-LPR with prominent DCS methods from the literature, whereby KNORA-U and META-DES are widely regarded as the current state of the art. This data confirms that KNORA-U is an outstanding dynamic classifier selector. Surprisingly, we find that simple majority voting is the runner-up to KU which clearly outperforms ACR, MCB, and OLA on these benchmarks.

Regarding CSHC-LPR, we see that it compares very favorably with all other methods. Even KNORA-U, the strongest competitor in terms of average accuracy, is outperformed on 24 out of 40 benchmarks. Looking at each fold separately, out of the 200 head-to-head comparisons, CSHC-LPR wins 121 and is tied 3 times with KU, which wins in 76 cases. Running an unpaired student t-test (likewise for Welch’s test) based on this win/loss data results in a p-value of less than $10^{-4}$ for the Null-hypothesis that both methods performed equally well, which allows us to refute this hypothesis with high statistical significance.

Furthermore, we can observe from this table that the original, unmodified CSHC method is almost as good as the best DCS methods to date. Compared with KNORA-U, it performs better on 15 benchmarks and worse on 24, with one benchmark tied. In practice, this makes CSHC a very attractive choice, since it does not require running all base classifiers, but only the one it selects. This means that an ensemble of classifiers can be used effectively for boosting accuracy without having to pay a significantly higher computational cost which may be attractive for keeping the energy and $\text{CO}_2$ burden low for mass applications.

VI. CONCLUSION

We offered three contributions: 1. For the first time in the literature, we showed that vanilla CSHC is a competitive DCS method. This finding is significant as it offers the possibility of achieving ensemble performance while only running one carefully selected classifier in the ensemble. No matter how many classifiers in the ensemble, the overhead is constant and only determined by running CSHC on top of the selected classifier. This is a huge advantage for mass applications and can help on the path for more environmentally sustainable AI. 2. We tailored CSHC for DCS by adding rank aggregation, an adaptive LP-based weighting scheme, and a recourse method, which resulted in a new DCS method we refer to as CSHC-LPR. 3. We provided extensive numerical results which showed that CSHC-LPR outperforms the state-of-the-art DCS methods with high statistical significance.

Finally, we also augmented CSHC-LPR to perform stacking and compared performances with other stacking approaches. Our results (not included for space restrictions) show that CSHC-LPR achieves almost identical performance as stacking, while allowing us to inherit justifications from base classifiers. Note that the same claim could not be made for the original CSHC method.

REFERENCES

[1] D. H. Wolpert, “Stacked Generalization,” Neural Networks, pp. 241–259, 1992.
[2] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for modern deep learning research,” in The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. AAAI Press, 2020, pp. 13 693–13 696. Available: https://ojs.aaai.org/index.php/AAAI/article/view/7123
[3] M. Abedini, N. C. F. Codella, J. H. Connell, R. Garnavi, M. Merler, S. Pankanti, J. R. Smith, and T. F. Syeda-Mahmood, “A generalized framework for medical image classification and recognition,” IBM J. Res. Dev., vol. 59, no. 2/3, 2015. [Online]. Available: https://doi.org/10.1147/JRD.2015.2390017
[4] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham, “A portfolio approach to algorithm select,” in Proceedings of the 18th international joint conference on Artificial intelligence, ser. IJCAI’03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Aug. 2003, pp. 1542–1543.
| TABLE I: Comparing the average accuracy over 5 cross-validation folds of CSHC and CSHC-LPR with other state-of-art methods. |
| --- | --- | --- | --- | --- | --- | --- | --- |
| APR | MCI | OLA | MV | MD | KU | CSHC | LPR |
| balance-* | 76.16 | 78.72 | 84.64 | 77.76 | 79.79 | 79.84 | 81.92 | 80.80 |
| bank-ma* | 72.50 | 73.36 | 76.08 | 81.64 | 76.18 | 81.34 | 78.09 | 80.30 |
| Bioresp* | 74.31 | 75.15 | 75.34 | 76.03 | 75.34 | 76.46 | 75.39 | 76.33 |
| breast-c* | 95.28 | 95.86 | 96.28 | 96.86 | 96.43 | 96.86 | 95.85 | 95.99 |
| cmc | 46.91 | 47.73 | 50.10 | 51.94 | 51.39 | 52.55 | 54.11 | 54.99 |
| citeese* | 88.02 | 85.96 | 85.93 | 91.11 | 90.95 | 91.02 | 89.26 | 90.93 |
| credit-ap* | 82.91 | 82.73 | 82.91 | 84.04 | 82.90 | 84.04 | 84.20 | 85.07 |
|credit-g | 73.20 | 74.40 | 78.10 | 84.60 | 72.60 | 75.80 | 74.10 | 75.00 |
| diabetes | 74.09 | 73.83 | 75.78 | 75.53 | 75.52 | 76.43 | 76.30 | 76.43 |
| eeg-* | 43.61 | 42.97 | 42.81 | 40.43 | 43.26 | 40.68 | 43.66 | 42.89 |
| eculaplys | 38.86 | 42.39 | 45.65 | 46.74 | 44.02 | 46.33 | 47.28 | 46.06 |
| gina_ap* | 85.61 | 87.14 | 86.59 | 88.15 | 88.90 | 88.67 | 89.53 | 89.62 |
| heart-* | 76.83 | 78.23 | 79.93 | 79.91 | 79.56 | 80.93 | 79.91 | 81.27 |
| hill-val* | 54.37 | 53.47 | 54.30 | 55.03 | 64.52 | 55.28 | 54.79 | 56.27 |
| idap* | 66.74 | 68.71 | 67.35 | 67.94 | 67.41 | 71.01 | 67.63 | 68.84 |
| isolet* | 90.23 | 91.82 | 93.55 | 94.50 | 94.64 | 94.83 | 95.15 | 95.14 |
| kc1 | 81.18 | 81.93 | 82.65 | 84.45 | 82.98 | 84.64 | 83.74 | 83.83 |
| kc2 | 77.97 | 79.10 | 81.21 | 82.55 | 81.21 | 82.74 | 82.34 | 83.31 |
| kr-vs-kp | 92.02 | 92.30 | 93.77 | 93.74 | 97.18 | 93.56 | 96.97 | 96.37 |
| letter | 92.40 | 92.21 | 92.54 | 91.29 | 93.11 | 93.66 | 93.72 | 93.94 |
| mfeat-fac* | 94.95 | 95.80 | 97.75 | 96.85 | 97.00 | 97.20 | 97.20 | 97.25 |
| mfeat-ion* | 79.35 | 80.83 | 79.05 | 82.00 | 81.05 | 81.83 | 81.80 | 82.15 |
| mfeat-melan* | 82.96 | 82.82 | 82.89 | 84.04 | 86.10 | 82.38 | 87.40 | 87.01 |
| mfeat-musk | 87.57 | 87.62 | 92.23 | 89.54 | 90.65 | 90.77 | 88.12 | 88.12 |
| mfeat-nokia | 92.20 | 92.30 | 92.79 | 93.57 | 93.58 | 93.75 | 93.52 | 93.91 |
| mfeat-speech | 95.60 | 95.71 | 96.00 | 97.35 | 97.44 | 97.38 | 97.54 | 97.63 |
| mfeat-ozf* | 92.80 | 92.34 | 93.09 | 93.41 | 93.84 | 93.57 | 93.29 | 93.57 |
| pc1 | 91.80 | 91.71 | 92.34 | 92.88 | 91.26 | 92.70 | 92.97 | 93.42 |
| pc2 | 87.50 | 88.04 | 88.93 | 89.51 | 89.25 | 89.38 | 88.87 | 89.25 |
| penning | 98.44 | 98.35 | 98.34 | 98.81 | 98.14 | 98.14 | 98.46 | 98.12 |
| pima | 88.14 | 87.45 | 86.62 | 84.92 | 87.81 | 86.35 | 86.45 | 86.58 |
| spambase | 83.41 | 83.31 | 84.83 | 85.78 | 85.40 | 87.11 | 84.46 | 86.35 |
| scene | 92.36 | 93.31 | 95.47 | 96.05 | 96.14 | 95.97 | 96.09 | 96.35 |
| speedi2l* | 76.02 | 77.11 | 78.52 | 83.37 | 80.40 | 83.29 | 84.29 | 84.32 |
| splice | 89.50 | 89.00 | 90.25 | 92.23 | 91.97 | 92.10 | 90.13 | 92.16 |
| tic-tac* | 75.28 | 74.76 | 72.98 | 74.04 | 76.74 | 75.27 | 73.92 | 76.74 |
| vehicle | 71.28 | 71.76 | 74.94 | 75.33 | 75.78 | 75.81 | 73.85 | 73.18 |
| vowel | 70.94 | 54.44 | 58.99 | 57.78 | 58.48 | 63.93 | 50.90 | 52.72 |
| wdbc | 95.96 | 95.25 | 97.02 | 97.54 | 97.01 | 97.19 | 98.14 | 97.01 |
| winaes/LPR | 38.00 | 37.00 | 33.00 | 28.00 | 29.00 | 24.00 | 32.00 | 0.00 |
| rank | 2.01 | 2.53 | 4.05 | 5.31 | 5.08 | 6.00 | 5.23 | 6.56 |
| winn/LPR | 2.60 | 3.00 | 7.00 | 12.00 | 11.00 | 16.00 | 7.00 | 0.00 |
| MGL [9] | 4.10 | 3.10 | 1.50 | 0.60 | 0.50 | 0.00 | 0.70 | 0.00 |
| μ accuracy | 79.88 | 80.53 | 81.72 | 82.40 | 82.45 | 82.85 | 82.55 | 82.90 |