Improving the Search of Learning Classifier Systems Through Interpretable Feature Clustering

Hayden Andersen
andershayd@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Andrew Lensen
andrew.lensen@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

Will N. Browne
will.browne@qut.edu.au
Queensland University of Technology
Brisbane, Australia

ABSTRACT

Learning Classifier Systems (LCS) are a well-known machine learning method, producing sets of interpretable rules in order to solve a variety of problems. Despite this, a common issue that these systems run into is the creation of unhelpful rules, caused by having multiple features in the data representing similar areas of knowledge. While we can logically know that these rules will not be useful in conjunction with each other, this is much more difficult for the algorithm to innately know.

This paper presents an exploration into using clustering algorithms for feature selection in LCS, selecting features that represent each major cluster of feature information. Combined with the innate power of LCS at finding nonlinear decision boundaries, these selected features can achieve results close to that of the full feature set while reducing the training time required to reach those results. The feature selection performed is highly interpretable, allowing for different features to be selected while maintaining the information spread in the feature subset.

ACM Reference Format:
Hayden Andersen, Andrew Lensen, and Will N. Browne. 2022. Improving the Search of Learning Classifier Systems Through Interpretable Feature Clustering. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3520304.3534027

1 INTRODUCTION

Learning Classifier Systems (LCS) [1] are an Evolutionary Computation method that learns a population of rules in order to perform tasks. One common application for LCS systems is classification, where each rule makes an individual mapping of the class from an input, and these heuristics are combined to form an overall class prediction.

However, LCS algorithms are known to spend a considerable amount of time in searching in "unpromising areas" where no good rules can be found [2]. This makes for an inefficient algorithm that ultimately produces rules which are less effective than potential alternatives.

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07…$15.00
https://doi.org/10.1145/3520304.3534027

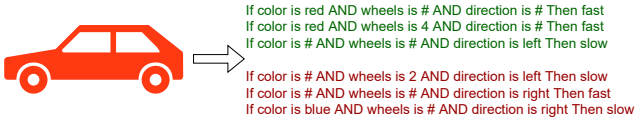

Figure 1: Example of an LCS rule set. Rules in green match the input, and rules in red do not.

Clustering is a form of machine learning that falls under the category of unsupervised learning [3], grouping similar instances of unlabelled data into distinct subsets known as clusters. This is performed without any knowledge about the data other than the data points themselves. Through slight modifications, clustering algorithms can be used to cluster features in the data, rather than instances. This has been used previously for feature selection, producing strong results [4].

This position paper explores the use of clustering algorithms as an interpretable algorithm to select features for LCS, addressing the problem of searching in unpromising areas. It is hypothesised that this will select features that are especially good at utilising the ability of LCS to find nonlinear decision boundaries, as each cluster will represent different areas of knowledge in the feature set.

2 BACKGROUND

2.1 Learning Classifier Systems

Learning Classifier Systems (LCS) [1] are a broad descriptor of a class of Evolutionary Computation algorithm that learns a population of models over a series of generations. For the purposes of this paper we consider only 'Michigan Style' LCS, which evolve a population of rules that are then considered as a whole entity when creating a prediction from the system. More precisely, this paper considers the sUpervised Classifier System (UCS) algorithm [5], which is a supervised learning LCS system that is designed to simplify the reward criteria when the correct action is always known immediately.

A rule in UCS is defined, in essence, as an "If *conditions* Then *action*" statement. The conditions are quantifiers over the values taken in the data and the action is the predicted class — for example, a rule could be "If color is red AND wheels is 4 Then fast". Not all features in the data have to be used in a rule. Conventionally, a feature that exists but is not counted in a rule is denoted as a '#' symbol in the rule, representing the fact that any value can fit that criteria. Discrete features are handled in a rule by denoting the exact value or a set of exact values of that feature, while continuous features are denoted with an upper and lower bound for the feature. Figure 1 shows an example of a basic LCS rule set.

The basic idea of UCS is to iterate through each instance in the dataset, evaluating if they fit any of the existing rules. A rule matches the data if all the conditions in the rule are met by the instance. From this, a match set of all such rules is formed. Each rule in the match set with the correct prediction then has their fitness increased, while rules with the incorrect prediction have their fitness decreased. If there are no matching rules, a new rule is created that matches the instance with the correct prediction, with added generality. After a set number of instances are seen by the system, the rules are modified through the genetic operations of crossover and mutation in order to find the optimal rules for the data.

LCS systems are known to be highly interpretable machine learning models, as the population of rules can be directly read by a user. They provide both global and local explanations with ease: the full set of rules can be explored to get a general idea of the model, and the specific rules that apply to a prediction can also be explored in order to provide a local explanation for that prediction.

2.2 Feature Selection

Feature selection [6] is a form of data transformation that selects a subset of the features in a dataset to use in further algorithms. There are many reasons to perform feature selection — it can improve the efficiency of machine learning models by requiring less features to predict on, it can improve performance by removing redundant or counterproductive features from the data, and it can improve interpretability by reducing the full feature set down to a human-understandable number of features.

There exist three main categories of feature selection algorithm: filter, wrapper, and embedded [7]. Filter methods perform feature selection as a separate task to the main machine learning problem, selecting features based only on the relationships in the data. Wrapper methods, on the other hand, utlising the machine learning algorithm within the feature selection to tailor the selected features to that algorithm. However, this can be very computationally expensive — especially for evolutionary algorithms such as LCS. Finally, embedded algorithms perform feature selection alongside the machine learning algorithm.

2.3 Clustering Algorithms

This paper explores the use of both partitional and density-based clustering algorithms. While these algorithms share some similarities, they differ in how clusters are defined and in the consideration of outliers.

There are two partitional clustering algorithms explored in this paper, k-means and k-medoids. The k-means algorithm first selects k random instances on a uniform distribution from the dataset, and sets them to be the centre of each cluster. Every instance is then added to the nearest cluster, and then the cluster centres are updated. This continues until the partitions between distinct clusters does not change [8]. The k-medoids algorithm is very similar, except the cluster centre is always set to be the central-most instance in a cluster rather than an arbitrary point in vector space [9]. An important distinction between these two algorithms is that the k-means algorithm defines arbitrary points in space to represent the centre of a cluster. In contrast, the k-medoids algorithm will always use an actual data point as the representation of the centre of a cluster. This allows for more flexibility, as any precomputed dissimilarity values can be provided to the algorithm.

Algorithm 1: Feature selection algorithm through feature clustering

Parameters: T training data, D dissimilarity function, C clustering algorithm

$F \leftarrow$ transpose of T;
$c \leftarrow$ clusterings of F using $C(F, D)$;
$features \leftarrow$ empty list;
for *cluster in c* **do**
  find most central value in *cluster*;
  add value to $features$;
end
return $features$;

The density based clustering method used in this paper is Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [10]. This is an extension of the DBSCAN algorithm, and creates clusters by finding a selection of core points for each cluster, then adding outer points to to complete the clustering. Unlike k-means and k-medoids, HDBSCAN does not need to be provided with the desired number of clusters and instead decides on how many clusters to create based on the amount of groups of core points. Additionally, this algorithm will potentially mark points as outliers, and not include them in any cluster. As with k-medoids, this algorithm only considers the relationship between defined features and therefore can be provided with precomputed dissimilarity values.

3 PROPOSED METHOD

The proposed method is to use clustering algorithms to select a good feature subset for use in LCS classification. 5 variations of this method are explored, using different combinations of clustering algorithms and dissimilarity measures. This is used as a filter feature selection method.

The data features are first clustered by applying a given clustering algorithm to the transposed dataset. This groups features into clusters, where each cluster contains features that represent similar information in the data. Next, the centre feature is selected from each cluster. This is chosen as it is assumed that the feature in the centre of each cluster will represent the most general information of that cluster. Each of the features not selected in this manner are permanently set as # in the LCS. Algorithm 1 shows a general overview of the algorithm used.

Figure 2 shows an example of clustering features in a dataset. In this example, the ten features $F_1, \ldots, F_{10}$ are grouped into three distinct clusters. Using these clusterings, three features in total would be chosen as part of the feature selection: F_1, F_5, and F_6, as these are the closest features to the centre of each cluster.

3.1 Dissimilarity Metrics

There are two different metrics of dissimilarity used in the clustering of features. The first is Euclidean distance, which is the distance between two features in a straight line (as the crow flies). The

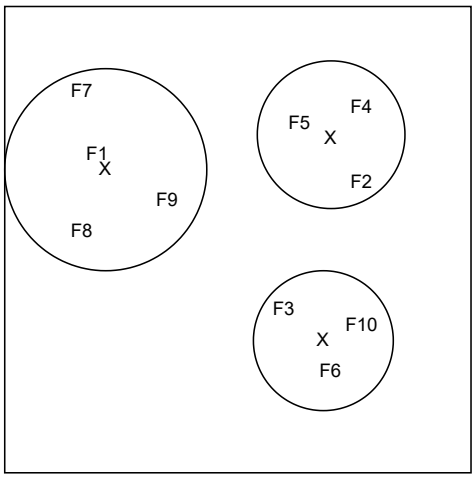

Figure 2: Example feature clusterings. The centre of each cluster is marked with an X

distance D is defined by Equation (1), where n is the number of instances in the dataset and F_{1i} and F_{2i} are the two features being compared on the ith instance in the dataset.

$$D = \sqrt{\sum_{i=1}^{n}(F_{1i} - F_{2i})^2} \tag{1}$$

The second similarity metric used in the method is the mutual information between the distributions of features. In order to treat this as a metric, the variation of information (VI) between features is taken. The variation of information is defined by Equation (2), where $H(F_i)$ is the entropy of feature F_i and $I(F_i, F_j)$ is the mutual information between F_i and F_j.

$$VI(F_1; F_2) = H(F_1) + H(F_2) - 2I(F_1, F_2) \tag{2}$$

Each of these similarity metrics provide different qualities to the clustering method. The Euclidean distance metric is a much more simple metric, and is the one often considered when clustering algorithms are created. It can be calculated between a feature and any given point in the data space, and so works with any clustering algorithm. However, it can only properly give the distance between ordinal features, as nominal data has no way of measuring the innate distance between each feature.

The variation of information metric, on the other hand, requires a sample of the distribution of each feature. Due to this, it cannot give the similarity between a feature and arbitrary points. As a metric, it determines similarity based on the distributions of each feature rather than the exact values of each instance. As such, the variation of information metric is more suited to the task of clustering features. In addition to this, the variation of information metric has no issues with the type of data — it works just as well with ordinal or nominal features, or even a mix of the two. One downside of this metric is that it can only be used for clustering algorithms that just evaluate distances between features and not algorithms that create arbitrary points. For example, the k-medoids algorithm must be used in place of the k-means algorithm.

Table 1: Experiment Parameters

Dataset	Pop. Size	Total Features	Features Selected (k)
Sonar	5000	60	10
Wine	1000	13	5
Zoo	150	17	6
Mushroom	1000	22	10

3.2 Clustering Algorithms

As the k-means algorithm utilises arbitrary points in space to represent the centres it can only be used with the Euclidean distance metric. However, as the k-medoids and HDBSCAN algorithms only use known data points, they can also utilise the variation of information metric between features.

As the clusters in HDBSCAN are not as spherical as those in the partitional clustering methods, a number of methods of determining the centre cluster were explored. The final decided method was to first calculate the geometric centre of each cluster, then to find the closest feature to each of those centres. Additionally, outlier features are chosen to be kept, as they represent information that is not contained by any of the clusters.

4 EXPERIMENTAL DESIGN

The proposed method is implemented in Python, utilising the basic eLCS library [1] as a baseline model of learning classifier systems. The clustering algorithms used are the Scikit learn implementations of k-means and k-medoids[11] and the original implementation of HDBSCAN [10]. The entropy and mutual information metrics for the variation of information are estimated using npeet [12]. [1]

Four datasets from the UCI machine learning repository [13] are evaluated — Wine, Sonar, Zoo, and Mushroom. These datasets were chosen to provide a range of problem types, with the Wine and Sonar datasets containing continuous, real valued data and the Zoo and Mushroom datasets containing categorical data. The Sonar and Mushroom datasets are both binary classification problems, while the Sonar and Zoo datasets are both multiclass classification problems.

Each experiment is repeated 30 times over different random seeds, in order to account for variation in the methods. In addition to this, due to the low number of instances in each dataset, each experiment performs 10-fold cross validation on the full dataset.

The experiments were run on a high performance computing grid in order to allow for more results to be computed at once.

Standard UCS parameters, as defined by Urbanowicz and Browne [1], were used for the LCS in all experiments. The one difference to this is the micropopulation size of the classifier system, as it was found in initial experimentation that some datasets require far less rules in order to achieve comparable performance. Each dataset also had a different number of set clusters for the k-means and k-medoids algorithms. These k values were found through initial experimentation, giving enough information to still successfully classify the data but representing a significant reduction in problem complexity. The chosen values for both of these parameters are shown in Table 1.

[1] Implementation code can be found at https://github.com/HaydenAndersen/ClusterLCS/

Table 2: Continuous data results (test set accuracy)

Method	Wine	Sonar
Baseline 1	0.965	0.658
Baseline 2	0.955	0.566
k-means	0.821	0.672
k-medoids	0.857	0.658
HDBSCAN	0.965	0.576
k-medoids VI	0.943	0.641
HDBSCAN VI	0.965	0.592

Table 3: Categorical data results (test set accuracy)

Method	Zoo	Mushroom
Baseline 1	0.929	0.946
Baseline 2	0.703	0.949
k-medoids VI	0.789	0.900
HDBSCAN VI	0.929	0.944

The experiments are run against two baseline models. The first baseline is simply using the full dataset as input to the eLCS system. The second baseline is a simple feature selection method that ranks all features in terms of entropy, then selects the top n features for selection. The idea behind this baseline is that it is based on information theory in the same way as the proposed method, and it is similarly an unsupervised filter feature selection method. This baseline is implemented using Scikit learn and the entropy is estimated using npeet. The number of features selected is the same as the number of clusters in the proposed method and the baseline experiments are run the same number of times and with the same random seeds. This provides the fairest comparison possible.

5 RESULTS

Table 2 shows the average accuracy of eLCS for each of methods on the Wine and Sonar datasets, and Table 3 shows the average accuracy of eLCS for the methods that work with categorical data on the Zoo and Mushroom datasets.

6 DISCUSSION AND ANALYSIS

While Baseline 1 does not perform any feature selection, it is still useful to demonstrate the possible performance of the classification algorithm when the full feature set is used. This allows us to evaluate how well the selected subset performs in comparison. However, the proposed methods are predominantly compared to Baseline 2, as another method of feature selection.

An interesting observation is that, at first glance, the HDBSCAN methods appear to perform as well as the full feature set. However, observing the clustering results shows that this is because the feature selection is actually selecting all features. A few small 'clusters' are created with only a single feature, and the rest of the features are marked as outliers. As the outliers are kept as selected features, this has the effect of not actually performing any feature selection. While this can be seen as the feature selection process considering every feature as important, the fact that it keeps every feature results in no improvement to the efficiency or performance of the LCS.

In terms of test set accuracy, the proposed method outperforms the baseline feature selection with all three partitional clustering methods on the Sonar dataset and the variation of information k-medoids method on the Zoo dataset. However, the baseline method outperforms the proposed method on the Mushroom dataset.

As a feature selection algorithm, this is a very understandable and interpretable method. As users we can easily visualise the clustering results, and therefore see which other features the selected feature represents in the final subset. This essentially provides a direct mapping from the original feature set to the selected features. If a specific feature in the selected subset is not performing well, we can then select a different feature from its original cluster, without changing which other features are selected. This will retain the overall spread of information in the subset, as features are selected from the same clusters.

A potential issue causing the proposed methods to not reach their full potential performance is that the feature chosen is the one that is closest to each cluster centre. However, these features themselves may not encode much information and so may not actually provide much value to the overall feature selection. An interesting extension to this method would be to choose the feature in each cluster that contains the highest entropy, or potentially weight the entropies by closeness to the cluster centres, and then choose the highest scored feature. This would have the effect of selecting what should be the most informative feature from each cluster, rather than the most representative feature.

7 CONCLUSIONS AND FUTURE WORK

This position paper has introduced a novel feature selection algorithm for use in LCS, with some promising results gathered from early experiments. The algorithm is very transparent in selecting certain features, meaning that when using it we can easily visualise and understand why specific features are being selected.

Despite this, further research is required in order to find the best combination of clustering algorithm and dissimilarity measure to use in order to produce the best possible results. This includes the exploration of different families of clustering algorithm, such as hierarchical clustering and graph-based clustering. Additionally, further research is required into which feature to select to best represent each overall cluster. Finally, further work is needed to explore the visualisation of the proposed feature selection method, in order to demonstrate the interpretability of the algorithm.

Using these feature clusters, it is also theoretically possible to cluster LCS rules throughout the training process. Rules that utilise features in the same cluster can be considered clustered together, and therefore be marked as similar. This would prevent rules that are too similar being combined, and help the system more easily discover the best rules for a specific problem.

REFERENCES

[1] R. J. Urbanowicz and W. N. Browne, *Introduction to Learning Classifier Systems*, ser. Springer Briefs in Intelligent Systems. Springer, 2017. [Online]. Available: https://doi.org/10.1007/978-3-662-55007-6
[2] Y. Liu, "Learning Classifier Systems for Understanding Patterns in Data," 6 2021.
[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer series in statistics, 2 2009.
[4] A. Gupta, A. Gupta, and K. Sharma, "Clustering based feature selection methods from fmri data for classification of cognitive states of the human brain," in *2016*

3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 3581–3584.

[5] E. Bernadó-Mansilla and J. M. Garrell-Guiu, "Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks," *Evol. Comput.*, vol. 11, no. 3, p. 209–238, sep 2003. [Online]. Available: https://doi.org/10.1162/106365603322365289

[6] M. Kuhn and K. Johnson, *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019.

[7] S. Biswas, M. Bordoloi, and B. Purkayastha, "Review on feature selection and classification using neuro-fuzzy approaches," *International Journal of Applied Evolutionary Computation*, vol. 7, pp. 28–44, 10 2016.

[8] J. MacQueen, "Some methods for classification and analysis of multivariate observations," 1967.

[9] X. Jin and J. Han, *K-Medoids Clustering*. Boston, MA: Springer US, 2010, pp. 564–565. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_426

[10] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining*, J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172.

[11] 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*, vol. 12, pp. 2825–2830, 2011.

[12] G. V. Steeg and A. Galstyan, "Non-parametric Entropy Estimation Toolbox (NPEET)," 2014. [Online]. Available: https://www.isi.edu/~gregv/npeet.html

[13] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml