An Effective Action Covering for Multi-label Learning Classifier Systems: A Graph-theoretic Approach

Shabnam Nazmi
North Carolina A&T State University
Greensboro, North Carolina
snazmi@aggies.ncat.edu

Abdollah Homaifar*
North Carolina A&T State University
Greensboro, North Carolina
homaifar@ncat.edu

Mohd Anwar
North Carolina A&T State University
Greensboro, North Carolina
manwar@ncat.edu

ABSTRACT

In Multi-label (ML) classification, each instance is associated with more than one class label. Incorporating the label correlations into the model is one of the increasingly studied areas in the last decade, mostly due to its potential in training more accurate predictive models and dealing with noisy/missing labels. Previously, multi-label learning classifier systems have been proposed that incorporate the high-order label correlations into the model through the label powerset (LP) technique. However, such a strategy cannot take advantage of the valuable statistical information in the label space to make more accurate inferences. Such information becomes even more crucial in ML image classification problems where the number of labels can be very large. In this paper, we propose a multi-label learning classifier system that leverages a structured representation for the labels through undirected graphs to utilize the label similarities when evolving rules. More specifically, we propose a novel scheme for covering classifier actions, as well as a method to calculate ML prediction arrays. The effectiveness of this method is demonstrated by experimenting on multiple benchmark datasets and comparing the results with multiple well-known ML classification algorithms.

CCS CONCEPTS

• **Computing methodologies** → **Rule learning**; **Structured outputs**; • **Mathematics of computing** → **Graph theory**.

ACM Reference Format:
Shabnam Nazmi, Abdollah Homaifar*, and Mohd Anwar. 2021. An Effective Action Covering for Multi-label Learning Classifier Systems: A Graph-theoretic Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference 2021 (GECCO '21).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449639.3459372

1 INTRODUCTION

In multi-label (ML) classification problems, each problem instance is annotated with more than one label at the same time. ML problems appear in various domains, such as image annotation [7], text categorization [26], and gene function analysis [11]. The objective of a ML classification algorithm is to predict a set of relevant labels for a given problem instance. One of the challenges with ML problems is that the size of the label space grows exponentially with the number of labels and exploiting the label correlation information is an effective strategy to address this challenge [44]. Over the past decade, the research on multi-label classification methods has been focused on utilizing dependencies among the labels using various techniques such as probabilistic networks, graph structures, and recurrent neural networks.

Learning classifier systems (LCSs) are adaptive rule-based machine learning algorithms that utilize evolutionary computation strategies such as mutation and crossover to explore the problem space. LCSs evolve a population of local sub-models that collectively explain the target problem and offer human-readable rules. Due to their minimal assumptions about the underlying regularities in the problem, LCSs have been successfully applied to problems in various domains, such as epidemiological data sets [35, 38], sentiment analysis of short texts [1], analyzing human decision making in agent-based social simulations [19], and multi-label classification problems [23]. Moreover, the flexible structure of the rules in LCSs allows for the condition and the action of rules to be adapted to the assumptions of a specific problem. For instance, the action of a rule in LCS can be adapted to support various abstraction methods, such as neural networks, tree representations, or graph structures.

The multi-label classifier system proposed in [22, 23], namely the classifier system with ML rules (MLR), employs the label powerset (LP) approach for covering new classifiers to capture the high-order label relations. More specifically, covering treats each distinct combination of observed labels in the training data as a unique label to be assigned to the action of a covered classifier. Nonetheless, covering the actions using the LP method makes minimal use of the label dependency information and ignores other potentially useful local or global information. To calculate the prediction array, MLR combines the predicted labelsets in the match set of an instance and calculates a score using the classifier fitness for each label, where fitness is a function of the Hamming loss of the classifier's action. Yet, in most of the multi-label classification problems, classifier actions have many labels and a scalar fitness value does not reflect the true predictive ability of the classifiers.

In this paper, we leverage the flexible design and generalization capabilities of the LCS algorithms in solving ML classification tasks and effectively training multi-label classifier models. We propose a novel similarity-guided covering scheme for the classifier *action* using label similarity graphs. Weighted undirected graphs are used to model the pair-wise label relations locally and extract highly correlated label subsets. The proposed covering scheme leverages

Corresponding author*.

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07…$15.00
https://doi.org/10.1145/3449639.3459372

label dependency information to assign accurate labels to the classifier action. Moreover, we employ a label-specific precision measure to assess the classifier's confidence in predicting each class label. We integrate this measure into the prediction array calculation and implement an error correction scheme to establish an implicit stratification between the relevant and irrelevant classes to the instances.

2 BACKGROUND

2.1 Learning classifier systems

LCSs are a family of machine learning algorithms that employ genetic algorithms (GAs) to train a rule-based model that describes the target problem through human interpretable rules [40] [36]. Accuracy-based LCS is the most popular LCS formulation which includes UCS [4] and XCS [40]. UCSs function under a supervised learning framework and use an accuracy-based fitness evaluation to guide the exploring process [21]. XCSs are reinforcement-based: they employ a Q-learning like algorithm to explore and exploit problem instances. These algorithms can address both single-step and multiple-step problems, including sequence labeling, sequential decision making, regression, and classification. More recent variants of the supervised learning classifier systems, namely ExSTraCSs, take advantage of heuristic components such as the EK-UCS [37], the UCS-AT, and UCS-AF [33] mechanisms. The EK-UCS is an expert-knowledge guided covering and mutation mechanism, while UCS-AT and UCS-AF are developed to explore complex attribute interactions within ExSTraCS.

LCSs have been adopted to both classification and regression problems by utilizing various *action* specifications. To model continuous end-point problems, LCSs using fuzzy logic were the first variant to produce a continuous output [6]. Later, XCS algorithm was extended to XCSF [41][42], which was able to perform a function approximation task through reward prediction. The XCS algorithm is also extended to learn code fragments as the action of the classifiers in [18]. Code fragments are tree expressions that are similar to the trees generated in genetic programming. In [34], the ExSTraCS algorithm was extended to develop the first supervised learning classifier to handle continuous end-point problems that incorporated interval prediction into the rules. NLCS [8] replaces the action part of a classifier in UCS with a simple neural network (NN) to obtain a more compact population and better generalization. Furthermore, authors in [20] developed the CN-LCS which explores various convolutional neural network (CNN) structures in combination with the UCS algorithm to examine the effectiveness of the neural-based learning classifier systems with deep feature maps. Finally, the MLR classifier system has been proposed in [23] that adapts the UCS algorithm to solve the multi-label classification problem by adapting the action of a classifier to predict a set of labels and introducing a fitness evaluation criterion based on the Hamming loss of a classifier's prediction.

2.2 LCSs for the multi-label classification problem

In most real-world multi-label problems, the labels are correlated and naturally co-occur. For instance, if an image is annotated with the labels "island" and "ocean," it is very likely that the label "boat" is also relevant to this image. Current studies focus on exploiting label correlations with various degrees [44]: the first-order strategy ignores the label correlations [7, 43], the second-order strategy handles the pairwise correlation among the labels [13, 17], and the high-order strategy considers the influence of all other labels on each label [5, 16, 24, 31]. For a review on ML classification methods, please refer to [14, 44].

LCSs were first adapted to ML problems in [39] by modifying the classifier action to learn binary-relevance (BR) vectors of the labels and evolve default hierarchies within multi-label data using organization classifier systems. Nonetheless, the BR approach to ML problems ignores the label correlations and may result in sub-optimal solutions in real-world settings. The classifier system with multi-label rules (MLR) [23] adopts the label-powerset approach to ML problems to handle the high-order label dependencies by extending the UCS algorithm. To adapt UCS to the ML problems, MLR replaces the action of a rule (r) with a set of labels ($\tilde{\mathbf{y}}$). Through an LP-based learning, the covering operator assigns the set of ground-truth labels of the training data to the action of its covered rules. The fitness of a classifier is calculated using the Hamming loss (hl) between the classifier action and the ground-truth labels of the samples it has matched. Fitness is calculate as follows.

$$\mathcal{F} = (1 - \frac{\sum hl}{exp})^{\nu}. \tag{1}$$

Here, exp is the classifier experience, and ν is a user-defined value. The authors in [23] compared the expected fitness of a classifier calculated using the Hamming loss metric ($\bar{\mathcal{F}}^{hl}$), and also using the exact-match ratio (em) of its action ($\bar{\mathcal{F}}^{em}$), showed that the inequality $\bar{\mathcal{F}}^{em} \leq \bar{\mathcal{F}}^{hl}$ holds for the average fitness. This inequality implies that the classifiers that are not accurate (i.e., $em \neq 0$) but partially predict the correct multi-labels, are considered contributing classifiers when fitness is based on Hamming loss. Such classifiers have a better chance of receiving a reproductive opportunity and remaining in the population and are shown to contribute to avoiding over-fitting in real-world problems.

One of the drawbacks of LP-based learning is that it cannot predict unseen label combinations in the test data. MLR addresses this challenge by employing an action aggregation for the classifiers in the match set (M). This approach considers the union of the predicted labels ($\bar{\mathbf{y}}$) in the match set, as shown in (2), and calculates the prediction array using the fitness ($\mathcal{F}$) and numerosity (num) of the classifiers in M. Therefore, the prediction value for the i^{th} label, f_i, is calculated as in (3).

$$\bar{\mathbf{y}} = \bigcup_{r \in M} \tilde{\mathbf{y}}^r, \tag{2}$$

$$f_i = \sum_{\substack{r \in M, \\ i^{th} label \in \tilde{\mathbf{y}}}} \mathcal{F}^r \times num^r. \tag{3}$$

The obtained prediction array provides a ranking for the predicted labels. This strategy is compared to the classical way of selecting the action provided by the fittest classifier in M and the experiments show that employing the aggregated approach leads to a better average performance on multiple benchmark datasets.

2.3 Notations

Let $\mathbb{X} \subset \mathbb{R}^d$ denote a feature space and $L = \{l_1, \ldots, l_m\}$ be a finite set of class labels. Each instance $\mathbf{x}_i \in \mathbb{X}$ is associated with a label vector $\mathbf{y}_i \in \{0,1\}^{m\times 1}$, where $y_{ij} = 1$ means l_j is relevant to $\mathbf{x}_i$, and 0 otherwise. Thus, $D = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ is a finite set of instances that are assumed to be randomly drawn from an unknown distribution and $Y = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]^T$ is the label matrix for D. Moreover, $F : \mathbb{R}^d \rightarrow \mathbb{R}^m$ is the multi-label predictor where, the predicted values can be regarded as the confidence of the model in the relevance of the predicted label. Here, $f(\mathbf{x}_i, l_j)$ is the predicted confidence of l_j for the i^{th} instance. Then, $h : \mathbb{R}^d \rightarrow \{0,1\}^m$ is the multi-label classifier which can be induced from F via a threshold function. In other words, $h_{ij} = [\![f(\mathbf{x}_i, l_j) > t(\mathbf{x}_i, l_j)]\!]$ uses a threshold function t and outputs 1 if the prediction is higher than t. We use the notation $Y_i = \{l_j | \mathbf{y}_{ij} = 1\}$ to denote the set of the relevant labels, and $\bar{Y}_i = \{l_j | \mathbf{y}_{ij} = 0\}$ to denote the set of the irrelevant labels to $\mathbf{x}_i$. Finally, we use $|\cdot|$ to denote the cardinality of a set and $[\![\omega]\!]$ returns 1 if the predicate ω holds, and 0 otherwise.

3 PROPOSED METHOD

In this section, we first explain the main learning framework of the multi-label LCS (MLCS) algorithm and then introduce an effective mechanism to incorporate label correlations into the classifier covering operator (discussed in section 3.1). Furthermore, we propose a new method to calculate the ML prediction array that integrates the label-specific classifier precision into the prediction array to reduce uncertainty in predicted labels, as discussed in section 3.2.

Like other LCS algorithms, MLCS starts with an empty population of rules and creates initial rules through covering, while GA is applied to P to search the prob space to maximize the fitness of the population. The ability of the covering operator to build the initial population of accurate rules is critical for implementing an efficient algorithm. The label similarities can be used to guide the covering operator to initialize the classifier action with highly correlated labelsets. Graph structures are able to capture pair-wise relations among the labels and can be used to decompose the labels of the training instances into highly correlated subsets. To focus on the locally shared correlations, the training data is decomposed into multiple disjoint clusters, and a distinct similarity graph is constructed for the labels in each cluster. This approach is consistent with the LCS's rule-based design that evolves local sub-models. Subsequently, MLCS evolves a separate population of rules for each cluster of the data. In the following, we provide basic definitions and then explain how the label graphs in MLCS are constructed.

Definition 3.1. Graph $g = (V, E, W)$ is an undirected weighted graph, with the vertex set $V = \{v_1, \ldots, v_q\}$ and $W = [w_{i,j}]_{q\times q}$, which is the positive weighted adjacency matrix of g. An edge e_{ij} exists between two vertices v_i and v_j, if $w_{i,j} > 0$.

Definition 3.2. A component of an undirected graph is an induced sub-graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the rest of the graph [25].

In this paper, the label space structure is represented with an undirected weighted graph g, where the vertices are a subset of labels ($V \subseteq L$), the edges E represents the co-occurrence of the labels observed in D, and the weight matrix W indicates the correlation between the pairs of labels. To build graphs that capture the local label similarities, the input space $\mathbb{X}$ is decomposed into c clusters as $D^k = (\mathbb{X}^k, Y^k)$ for $1 \le k \le c$, such that

$$D^k = \{(\mathbf{x}_i, \mathbf{y}_i) | \mathbf{x}_i \in \mathbb{X}^k, 1 \le i \le n^k\}, \quad (4)$$
$$Y^k = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{n^k}]^T.$$

For the k^{th} cluster of the data, MLCS calculates a distinct label graph g^k. The edge weight matrix W^k is calculated using the Cosine similarity between the label vectors. Assume $Y_{i.}$ and $Y_{j.}$ to be the i^{th} and j^{th} columns of Y^k, respectively. g^k is constructed as follows.

$$V^k = \{l_j | \exists y_{ij} = 1, 1 \le i \le n^k\}, \quad (5)$$

$$W^k = \begin{cases} w_{ij} & w_{ij} \ge \delta \\ 0 & o.w. \end{cases} \qquad w_{ij} = \frac{< Y_{i.}, Y_{j.} >}{\|Y_{i.}\| \|Y_{j.}\|}, 1 \le i, j \le |V^k|, \quad (6)$$

$$E^k = \{e_{ij} | \forall w_{ij} \neq 0\}. \quad (7)$$

Here, $< \cdot, \cdot >$ denotes the inner product of two vectors, and $\|\cdot\|$ denotes the l_2 norm of the vector. Applying the minimum similarity threshold δ in (6) leads to a sparse graph that contains only the label pairs whose correlation is larger than δ. A sparse graph representation can remove the semantically unrelated links to avoid the propagation of incorrect information [29]. The connected components (CC) of g^k comprises of the groups of labels that are highly correlated and are used by the proposed ML covering scheme to initialize the classifier's action. The induced sub-graphs of g^k with respect to the minimum similarity threshold δ is defined as follows.

$$g^k_{(cc,\delta)} = connected - components(g^k, \delta) \quad (8)$$

The connected components of a graph can be extracted by analyzing the matrix properties of the graph, such as the graph Laplacian, and using algorithms such as the Reverse Cuthill-McKee algorithm [2]. This structure is directly impacted by the threshold δ and determines the number of the classifiers which are covered for each training instance. Therefore, it impacts the rate at which the population size increases and should be tuned accordingly and with respect to the similarity information calculated for a given problem. To demonstrate how MLCS extracts the local dependency graphs of a dataset, the similarity graphs for the PASCAL-VOC6 dataset [12] are presented in Figure 1. Considering the similarity matrix of the dataset in Figure 1-(a), $\delta = 0.1$ is assumed to ensure that highly correlated label pairs will be preserved while covering the classifier's action. The dataset is clustered into $c = 5$ clusters, and label similarities are calculated for each cluster as shown in Figure 1- (b, c). The sparse structure of the similarity graphs in the clustered data shows how some labels are weakly related at the local level, e.g., *bus* and *car* labels in Figure 1 - (b), and some labels are strongly related, e.g., *person* and *bus* in Figure 1 - (c). Utilizing this information to guide the classifier covering can facilitate the training of the LCS by creating rules that are initialized with accurate actions.

Algorithm 1 summarizes the major steps of the training in the MLCS algorithm. During the training of MLCS, covering is activated under two conditions: (a) when the set of the matching classifiers is empty, i.e., $M = \emptyset$, (b) when the target labels are not completely covered by the matching classifiers, i.e., $\mathbf{y} \nsubseteq \bigcup_{r\in M} \tilde{\mathbf{y}}$. The second

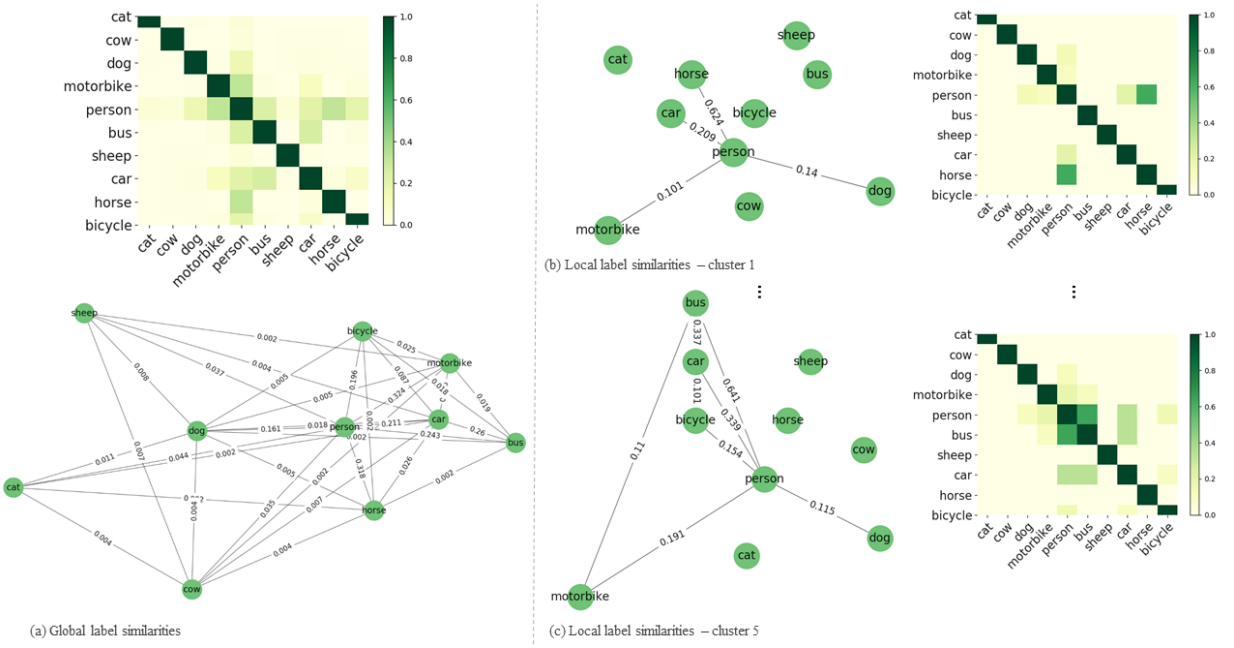

Figure 1: Visualization of the global and local label similarity graphs for the labels in PASCAL-VOC6 dataset.

condition increases the prediction capability of the model by creating new classifiers to cover the labels that are not present in M. The correct set (C) in MLCS comprises of classifiers whose predictions are a subset of the target labelset. This set is then used by the GA to randomly select parent classifiers, apply genetic operators on the classifier condition and create new off-springs. To evaluate the predictive performance of a trained population of rules, the ML-Predict algorithm is used (see section 3.2).

Algorithm 1 The pseudocode of the training algorithm of MLCS.

procedure TRAINING(D_{train}, δ)
  $(D^1, \ldots, D^c) \leftarrow$ cluster(D_{train}) (4);
  for $1 \leq k \leq c$ **do**:
    Calculate $g^k_{(cc,\delta)}$ according to (5) and (8);
  end for
  for $1 \leq k \leq c$ **do**:
    $it \leftarrow 0$;
    while $it \leq it_{max}$ **do**
      Read $(\mathbf{x}, \mathbf{y}) \in D^k$;
      Scan P^k and form match set M^k;
      if $(M^k = \emptyset) \mid (\mathbf{y} \nsubseteq \bigcup_{r \in M^k} \tilde{\mathbf{y}})$ **then**:
        ML-COVER($g^k_{(cc,\delta)}$, $(\mathbf{x}, \mathbf{y})$);
      end if
      Update rule parameters in M according to (1);
      Form correct set C^k and apply genetic algorithm;
      $it \leftarrow it + 1$;
    end while
  end for
  return $\{P^1, \ldots, P^k\}$;
end procedure

3.1 Multi-label action covering

In this section, we discuss the proposed ML covering strategy for the classifiers' action. The objective is to find label subsets that are highly correlated to create classifiers with maximally accurate actions. Given a training instance $(\mathbf{x}, \mathbf{y})$ that belongs to D^k, the covering operator obtains a partitioning of the labelset Y according to the label graph $g^k_{(cc,\delta)}$, as follows.

$$\{Y_1, Y_2, \ldots, Y_{q'}\} = g^k_{(cc,\delta)}(Y). \tag{9}$$

The labels that appear in each subset (Y_j) are assumed to be more similar than those in Y and will provide the initial population of rules with more accurate classifiers. For each labelset in (9), covering generates a new classifier and assigns the labelset to the classifier's action. To encourage variance in the initial classifiers inserted into the population, a unique condition is generated for each subset of labels that matches $\mathbf{x}$. Subsequently, covering creates at least one new classifier every time it is activated. For example, assume $Y = \{person, bus, car\}$ to be the labelset of a sample that belongs to D^1 of the PASCAL-VOC6 dataset. Covering decomposes Y into $Y_1 = \{bus\}$ and $Y_2 = \{person, car\}$, according to the label graph of this cluster (Figure 1 - (b)). Therefore, two classifiers are added to the population with distinct actions and random conditions. Algorithm 2 outlines the proposed covering scheme.

Algorithm 2 The pseudocode of the proposed multi-label covering.

procedure ML-COVER($g^k_{(cc,\delta)}$, $(\mathbf{x}, \mathbf{y})$)
  $V, E, W \leftarrow g^k_{(cc,\delta)}$;
  $Y \leftarrow$ relevant labels according to $\mathbf{y}$;
  $Y_1, Y_2, \ldots, Y_{q'} \leftarrow Y$ partitioned according to V;
  for $1 \leq i \leq q'$ **do**
    $\tilde{\mathbf{x}} \leftarrow$ random condition matching $\mathbf{x}$;
    $\tilde{\mathbf{y}} \leftarrow Y_i$;
    Insert $r = (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ into P^k;
  end for
end procedure

True labels: horse, person

Predicted Label	Confidence
person	1.0
sheep	0.6147
horse	0.5600
cow	0.4510
cat	0.3708

True labels: dog, person, motorbike

Predicted Label	Confidence
motorbike	1.0
person	0.9398
bicycle	0.8776
dog	0.8769
car	0.7986

True labels: bicycle, person

Predicted Label	Confidence
bicycle	1.0
person	0.914
motorbike	0.4903
horse	0.3677
cat	0.2826

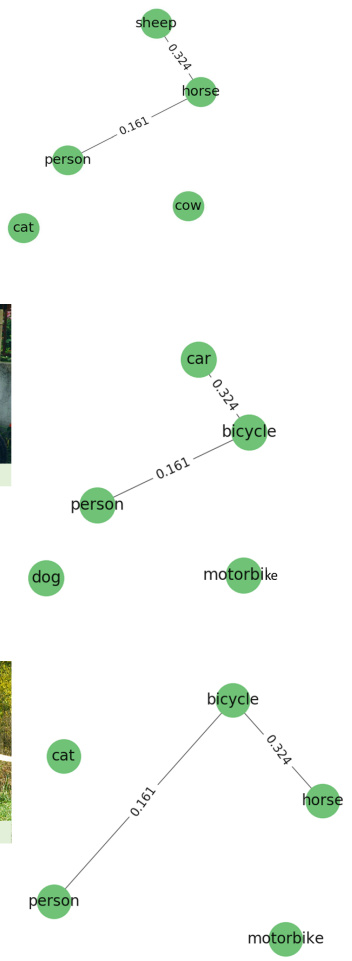

Figure 2: Sample outputs of a trained MLCS model on test images selected from the PASCAL-VOC6 dataset.

3.2 Multi-label prediction array

This section discusses the proposed strategy for calculating the multi-label prediction array, that address the drawback with using the fitness values in MLR. We introduce a label-specific precision measure (α) for a classifier that evaluates its confidence in predicting each label in its action and is calculated with respect to the ground-truth labels of the training instances it matches. For $l_j \in \tilde{\mathbf{y}}$, the α_j is derived as follows.

$$\alpha_j = \frac{\sum_{i=1}^{|M_r|} y_{ij}}{|M_r|}. \tag{10}$$

Here, M_r is the set of the training instances that r matches. The value of α_j also indicates the probability of observing a sample within the sub-space covered by $\tilde{\mathbf{x}}$ that belongs to the j^{th} class. To calculate the multi-label prediction array for a given instance $\mathbf{x}$, the instance is first assigned to its closest cluster D^k and the set of its matching classifiers are identified from P^k. Similar to MLR, we first obtain the union of all the predicted labels ($\bar{\mathbf{y}}$) in the match set to increase the generalization of the model to unseen label combinations.

$$\bar{\mathbf{y}} = \bigcup_{r \in M} \tilde{\mathbf{y}}. \tag{11}$$

For each label in $\bar{\mathbf{y}}$, a prediction value is calculated using the precision of the classifier that is scaled with respect to the number of classifiers predicting the same label. Let α_{ij} to be the precision of r_i in predicting l_j, and n_j to be the ratio of the number of the rules in M that has l_j in their action. The prediction value for this label is calculated using a sigmoid function as follows.

$$f(\mathbf{x}, l_j) = \frac{1}{1 + e^{-(n_j - \varepsilon)}} \max_{r_i \in M} \alpha_{ij}. \tag{12}$$

The prediction array calculated by (12), focuses on the labels that are most frequently predicted and belong to the most certain classifiers. Once all the scores are obtained for the samples in D^k, we obtain the ML binary classification vector h for each sample by applying a label-specific threshold value to the prediction array such that, for $\mathbf{x}$ and label l_j, this value is calculated as

$$h_j = [\![f(\mathbf{x}, l_j) > t(\mathbf{x}, l_j)]\!]. \tag{13}$$

Here, $t(\mathbf{x}, l_j)$ is a label-specific threshold that is calculated by maximizing the area under the precision-recall curve ($PR - AUC(f)$) of the predicted scores for the instances within a cluster, as follows.

$$t(\mathbf{x}, l_j) = \arg \max_{t \in [0,1]} PR - AUC(f(\mathbf{x}, l_j), t). \tag{14}$$

Algorithm 3 The pseudocode of the proposed multi-label covering.

```
procedure ML-PREDICT(X_test)
    for x_t ∈ X_test do
        Identify D^k and P^k;
        M ← match set of x_t from P^k;
        Calculate ȳ using (11);
        for l_j ∈ ȳ do
            Calculate f(x_t, l_j) using (12);
        end for
    end for
    Calculate thresholds t_j^k, 1 ≤ j ≤ m, 1 ≤ k ≤ c using (14);
    for x_t ∈ X_test do
        Obtain ML prediction h using (13);
    end for
end procedure
```

4 RESULTS AND DISCUSSION

In this section, the benchmark datasets and several classification algorithms that are used in the comparison experiments are described. Then, multi-label evaluation measures are explained, and the strategies employed for parameter instantiation are reported. Finally, results are presented and discussed.

4.1 Benchmark Datasets

For comparison, five popular real-world datasets are used including, PASCAL-VOC6, Scene [7], Mediamill [28], Corel5k [10], and Corel16k [3]. Table 1 shows the information about the number of instances, features, and classes for each dataset. The raw images for the PASCAL-VOC6 dataset are available at *PASCAL Visual Object*

Dataset	Sample count	Feature count	Label count	*Card*	*Dens*
PASCAL-VOC6	3,500	256	10	1.309	0.131
Scene	2,407	294	6	1.074	0.179
Mediamill	43,907	120	101	4.555	0.045
Corel5k	5,000	499	374	3.522	0.009
Corel16k	13,770	500	153	2.859	0.019

Table 1: ML datasets used in the comparative experiments.

Algorithms	Parameters	Values
MLR and MLCS	it_{max}	100,000
	θ_{GA}	15
	p_χ	0.8
	p_μ	0.05
	ν	1
MLCS	c	5
	ε	0.5

Table 2: The LCS algorithm parameter settings.

Classes [1] home page. We have utilized a pre-trained convolutional neural network, VGG16 model [27], to extract a 256-dimensional feature map for each image and used a 70-30 percent split of the data to train and test the algorithms. For other datasets, the train-test splits are available at the *MULAN* [2] library. For each dataset, the label cardinality (*Card*) is the average number of labels per sample, and label density (*Dens*) is the cardinality divided by the number of labels [44].

4.2 Experimental setup

To demonstrate the improvement obtained from the proposed ML covering and prediction strategies, the MLR algorithm [23] is included in the comparative experiments. To compare the overall predictive performance of MLCS, the multi-label k-nearest neighbors (ML-kNN) [43] is selected that ignores the label correlation information. Calibrated label ranking (CLR) [13] exploits the pairwise label correlations. The ensemble of classifier chaining (ECC) [24] and RAkEL [31] both utilize the high-order label correlations. Finally, hierarchy of multi-label classifiers (HOMER) [30] with a LP-based classifier trained at each node, leverages the high-order label correlations. In the experiments, whenever necessary, the decision tree classifier is opted for as the base learner. The experiments are performed using the implementations of the above multi-label classification algorithms in *MULAN* library under the machine learning framework *WEKA* [15]. The implementation of MLR in Python is available on GitHub [3]. MLCS is also implemented in Python and in available on GitHub [4]. All experiments are carried out on a 2.70 GHz Windows 10 machine with a 16.0 GB RAM.

4.3 Evaluation metrics

The performance of the compared ML classification methods are reported and compared in terms of the following ML classification evaluation metrics [21]. The *Average accuracy* metric is an example-based metric, while F_{macro} is a label-based measure and $Rank-basedprecision$ evaluates the performance based on the ranking of the predicted labels.

• *Average Accuracy* is the average relative number of classes predicted correctly to the set of all predicted classes over all examples.

$$Acc_{example}(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{m} y_{ij}h_{ij}}{\sum_{j=1}^{m} y_{ij} + \sum_{j=1}^{m} h_{ij}}$$

• *Macro-averaged F-score* is the harmonic mean between the macro-averaged precision and recall averaged over all labels.

$$F_{macro} = \frac{1}{m}\sum_{j=1}^{m}\frac{2\sum_{i=1}^{n} y_{ij}h_{ij}}{\sum_{i=1}^{n} y_{ij} + \sum_{i=1}^{n} h_{ij}}$$

• *Rank-based precision* is the average fraction of relevant labels ranked higher than one other relevant label.

$$Rank - Pr(F) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|Y_i|}\sum_{\lambda\in L_j}\frac{|S_{Pr}^{ij}|}{f(\mathbf{x}_i, l_j)},$$

where,

$$S_{Pr}^{ij} = \{k \in Y_i | f(\mathbf{x}_i, l_k) \leq f(\mathbf{x}_i, l_j)\}.$$

4.4 Hyper-parameter tuning

The hyper-parameters of the methods used in the for comparison are decided following the recommendations from the literature. In cases where a parameter is to be determined from a set of values, the value that corresponds to the maximum F_{macro} measure on each dataset is considered in the experiments. The number of models in RAkEL is set to $min(2m, 100)$ for all datasets [32]. The size of the labelsets for RAkEL is set to $m/2$ as it provides a balance between computational complexity and performance [24, 32]. On datasets with large number of labels this value is set to 3 to avoid memory error. The number of neighbors in the ML-kNN method for each dataset is selected from the set $(6, 20)$ with a step size of 2. The number of models in ECC is also set to 10 to be consistent with other ensemble methods. HOMER requires the number of clusters to be determined which is selected from $(2, 6)$ [30]. For the MLCS algorithm, as well as the MLR algorithm, the maximum number of classifiers, is selected between $(1000, 10000)$ by 1000 steps, and $P_\#$ is selected from $[0.1, 0.9]$, with a step size of 0.05. In MLCS, the δ values of the label similarities for the edge weights are selected from the $[0, 1]$ range. The remaining of the training parameters concerning the LCS algorithms are set as listed in Table 2.

4.5 Results and discussion

The performance of the proposed MLCS algorithm is investigated and reported from two aspects. The first set of results analyzes the impact of including each one of the proposed improvements in the MLCS algorithm and compares them with that of the MLR algorithm. These results are presented in Table 3 in terms of three evaluation metrics described in section 4.3. The values corresponding to each dataset and each method, are averaged over ten runs of the LCS algorithm and the mean values are reported. The entries

[1] http://host.robots.ox.ac.uk/pascal/VOC/databases.html
[2] http://mulan.sourceforge.net/
[3] https://github.com/ConflictedPhilosopher/MLR.git
[4] https://github.com/ConflictedPhilosopher/RELoC-GP.git

that are referred to as MLCS-I, demonstrate the impact of including the proposed covering method, while those referred to as MLCS-II present the impact of including both the covering and the new prediction array methods.

Comparing the results of MLR and MLCS-I in terms of the *average accuracy* shows that on four datasets the mean performance values have improved. This means that on average more of the relevant labels are recovered and fewer irrelevant labels are predicted. This can be explained by the way that the initial classifiers are covered. Initializing rules with the LP method in MLR leads to classifiers having more labels in their *action* on average compared to MLCS-I. Some of these labels may not co-occur on all of the instances that a rule matches. In MLCS-I, the covering operator breaks down the labelsets of the ground-truth data into more correlated label subsets. Therefore, a classifier's action is more likely to be correct on the samples it matches and have a smaller Hamming loss and a larger fitness value. Therefore, initializing the population with potentially more useful classifiers improves the overall predictive performance of the algorithm. A similar improvement is observed in terms of the F_{macro} measure, which evaluates the average performance of the model on individual classes. Creating separate classifiers to cover the labels that do not co-occur often enough, provides an opportunity for the classifiers to appear in the correct set more often. These accurate classifiers will have a better reproductive opportunity and remain in the population. Since in both of these experiments the prediction array is calculated using the fitness values, the performance of the algorithms in terms of the *rank precision* metric is directly affected by the classifier fitness. The better performance of the MLCS-I algorithm in terms of this metric shows that the algorithm was able to populate the rule base with better classifiers than MLR.

Comparing the results of MLCS-I and MLCS-II in terms of the three evaluation metrics, as presented in Table 3, shows that the average performance of the algorithm has improved on all datasets. Unlike using the Hamming loss to form the prediction array, the proposed variant in equation (12) uses the label-specific precision values and provides a more rigorous insight into the predictive capability of the model. This approach takes into account the probability of each class being observed as reported by the classifiers and focuses on the maximum probability in order to reduce the uncertainty caused by less certain classifiers. Then, the probability is scaled with the number of classifiers predicting a label non-linearly to prioritize the labels that are predicted by more classifiers and enforce an implicit stratification between the ranking of the relevant and irrelevant labels. The Wilcoxon signed-rank test [9] is employed to assess the statistical significance of the observed improvement (reported in Table 3). With a significance value of 0.05, the improvement gained from each new component is significant in terms of all metrics, excluding the *rank precision* in the experiments on MLCS-II.

The second set of results compares the performance of MLCS with the well-known ML classification approaches from the literature. The experiments are performed on five benchmark datasets and the mean value of the performance metrics are reported in Tables 4-6. For each method, their relative rank on a dataset is shown next to their metric values within the parentheses, and their average relative rank can be found under *Avg. rank* column. Finally, for each evaluation metric, the average rank of the method corresponding to the highest rank is printed in boldface. According to Tables 4-6, the proposed MLCS algorithm has the highest average relative rank in terms of all three evaluation measures, while it presents slightly worse performance than other methods in a number of scenarios. To assess the statistical significance of the observed differences in the performances, the Friedman statistic [9] is calculated for the average relative ranks of each method, as presented in Table 7. The test is performed for six methods experimented on five datasets and the significance level is set to be 0.05. According to the test results, the performance of the compared methods is not significantly different in terms of their *average accuracy* and F_{macro} measures, while their difference is significant in terms of the *rank precision*. Figure 3 presents the result of the Bonferroni-Dunn test to evaluate the critical distance between the average ranks for the *rank precision* metric, which shows that the performance of the MLCS is significantly better than that of the HOMER and RA*k*EL.

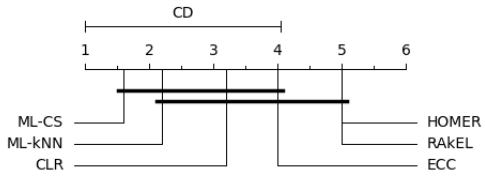

Figure 3: Comparison of the *rank precision* of MLCS algorithm against other ML classification methods with the Bonferroni-Dunn test and $\alpha = 0.05$.

To demonstrate the output of a trained MLCS model for a given sample, we have displayed three images from the PASCAL-VOC6 dataset in Figure 2, along with their true labels. For each image, their top five ranked labels are presented with their respective prediction values. The label similarity graph, constructed from the classifier actions in the match set of the sample, is shown to the right of each image. The edge weights express the Cosine similarity between labels, and the lack of an edge denotes a lower than δ similarity between labels. With the knowledge of the cluster that a sample belongs, the similarity graph for each image in Figure 2 is a subset of the similarity graph for their corresponding cluster in Figure 1.

5 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we proposed novel strategies for covering the action of a multi-label classification rule and to form a multi-label prediction array for ML learning classifier systems. The covering scheme exploits the label correlation information to initialize classifiers with subsets of correlated labels using locally induced undirected weighted graphs. The prediction array utilizes label-specific precision values of the classifiers and incorporates the size of the predicting classifiers into the function to calculate confidence values for each predicted label. The effectiveness of the proposed components are first investigated in an incremental manner and are compared to a base-line multi-label LCS. Furthermore, the performance of the proposed learning algorithm is compared to multiple well-known multi-label classification methods on multiple datasets

Datasets	PASCAL-VOC6	Scene	Mediamill	Corel5k	Corel16k	Wilcoxon test (w, p)
Average accuracy						
MLR	0.5725	0.5621	0.1074	0.0114	0.0147	
MLCS-I	0.5914	0.5576	0.1388	0.0224	0.0192	(256.0, 0.001)
MLCS-II	0.6949	0.6638	0.1887	0.0503	0.0314	(413.0, 0.030)
F_{macro}						
MLR	0.6682	0.6499	0.1251	0.2556	0.0085	
MLCS-I	0.7419	0.7114	0.1389	0.2776	0.0208	(267.0, 0.001)
MLCS-II	0.7463	0.7331	0.1566	0.3072	0.0294	(124.0, 0.0)
Rank precision						
MLR	0.7542	0.7588	0.6276	0.2035	0.2407	
MLCS-I	0.7991	0.8006	0.6857	0.2634	0.2819	(330.0, 0.043)
MLCS-II	0.8337	0.8419	0.7119	0.2864	0.3087	(381.0, 0.051)

Table 3: Performance comparison of MLCS and MLR algorithms. MLCS-I indicates the case where only $ML - Cover$ is utilized, and MLCS-II is when both $ML - Cover$ and $ML - Predict$ are used. w is the calculated test statistic and p is the probability value.

Datasets	PASCAL-VOC6	Scene	Mediamill	Corel5k	Corel16k	Avg. rank
RAkEL	0.6159(4)	0.5810(4)	0.3109(5)	0.0446(4)	0.0328(3)	4
ML-kNN	0.6945(2)	0.6614(2)	0.4219(2)	0.0140(6)	0.0106(6)	6
HOMER	0.5985(5)	0.5701(5)	0.3377(4)	0.1042(1)	0.0766(1)	3.2
ECC	0.6891(3)	0.6501(3)	0.4342(1)	0.0475(3)	0.0230(5)	3
CLR	0.5335(6)	0.5033(6)	0.4164(3)	0.0313(5)	0.0331(2)	4.4
MLCS	0.6949(1)	0.6638(1)	0.1887(6)	0.0503(2)	0.0314(4)	**2.8**

Table 4: The performance of the ML algorithms in terms of *Average Accuracy* ↑.

Datasets	PASCAL-VOC6	Scene	Mediamill	Corel5k	Corel16k	Avg. rank
RAkEL	0.7287(4)	0.6692(4)	0.1481(2)	0.3092(2)	0.0303(2)	2.8
ML-kNN	0.8220(1)	0.7189(2)	0.1081(5)	0.3066(4)	0.0148(5)	3.4
HOMER	0.7398(3)	0.5884(6)	0.1286(3)	0.2577(6)	0.0646(1)	3.8
ECC	0.6606(5)	0.7181(3)	0.1099(4)	0.3121(1)	0.0218(4)	3.4
CLR	0.6342(6)	0.6228(5)	0.0974(6)	0.3004(5)	0.0136(6)	5.6
MLCS	0.7463(2)	0.7331(1)	0.1566(1)	0.3072(3)	0.0294(3)	**2**

Table 5: The performance of the ML algorithms in terms of F_{macro} ↑.

Datasets	PASCAL-VOC6	Scene	Mediamill	Corel5k	Corel16k	Avg. rank
RAkEL	0.6908(4)	0.8162(4)	0.4847(5)	0.0586(6)	0.1011(6)	5
ML-kNN	0.8439(1)	0.8513(1)	0.7030(3)	0.2656(3)	0.2667(3)	2.2
HOMER	0.8101(3)	0.7139(6)	0.4158(6)	0.1197(5)	0.1135(5)	5
ECC	0.6473(5)	0.8389(3)	0.6864(4)	0.1729(4)	0.1764(4)	4
CLR	0.6214(6)	0.8128(5)	0.7230(1)	0.2743(2)	0.2997(2)	3.2
MLCS	0.8337(2)	0.8419(2)	0.7119(2)	0.2864(1)	0.3087(1)	**1.6**

Table 6: The performance of the ML algorithms in terms of *rank precision* ↑.

Evaluation metric	$\mathcal{F}_F$	p_{value}	Critical value
Average Accuracy	2.714	0.744	
F_{macro}	10.371	0.065	10.49
Rank-precision	14.486	0.013	

Table 7: Summary of the Friedman rank test for $\mathcal{F}_F(k = 6, N = 5)$ and $\alpha = 0.05$.

and the results are reported in terms of three evaluation metrics. The experiments and the accompanied statistical tests demonstrate the efficacy of the proposed components for the MLCS algorithm as well as its competing performance compared to the other classification methods.

In the future, authors will investigate the impact of the minimum label similarity threshold on the performance of the classifier system and the population size. In the current work, this threshold is treated as a hyper-parameter which is tuned along with the population size. Further research may analyze the sensitivity of the algorithm to this parameter.

ACKNOWLEDGMENTS

This work is supported by Air Force Research Laboratory and Office of Secretary of Defense under agreement number FA8750-15-2-0116 and Lockheed Martin under fund number 234363. The corresponding author would like to also acknowledge the NASA University Leadership Initiative under grant number 80NSSC20M0161. Also, this work is partially supported by the OUSD(RE)/RTL and was accomplished under Cooperative Agreement Number W911NF-20-2-0261.

REFERENCES

[1] Muhammad Hassan Arif, Jianxin Li, Muhammad Iqbal, and Kaixu Liu. 2018. Sentiment analysis and spam detection in short informal text using learning classifier systems. *Soft Computing* 22, 21 (2018), 7281–7291.

[2] Ariful Azad, Mathias Jacquelin, Aydin Buluç, and Esmond G Ng. 2017. The reverse Cuthill-McKee algorithm in distributed-memory. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 22–31.

[3] Kobus Barnard, Pinar Duygulu, David Forsyth, Nando De Freitas, David M Blei, and Michael I Jordan. 2003. Matching words and pictures. (2003).

[4] Ester Bernadó-Mansilla and Josep M Garrell-Guiu. 2003. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary computation* 11, 3 (2003), 209–238.

[5] Wei Bi and James T Kwok. 2014. Multilabel classification with label correlations and missing labels. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

[6] Andrea Bonarini. 1999. An introduction to learning fuzzy classifier systems. In *International Workshop on Learning Classifier Systems*. Springer, 83–104.

[7] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. 2004. Learning multi-label scene classification. *Pattern recognition* 37, 9 (2004), 1757–1771.

[8] Hai H Dam, Hussein A Abbass, Chris Lokan, and Xin Yao. 2007. Neural-based learning classifier systems. *IEEE Transactions on Knowledge and Data Engineering* 20, 1 (2007), 26–39.

[9] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[10] Pinar Duygulu, Kobus Barnard, Joao FG de Freitas, and David A Forsyth. 2002. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *European conference on computer vision*. Springer, 97–112.

[11] André Elisseeff and Jason Weston. 2002. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*. 681–687.

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2007. The PASCAL visual object classes challenge 2007 (VOC2007) results. (2007).

[13] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. 2008. Multilabel classification via calibrated label ranking. *Machine learning* 73, 2 (2008), 133–153.

[14] Eva Gibaja and Sebastián Ventura. 2015. A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)* 47, 3 (2015), 1–38.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.

[16] Zhi-Fen He, Ming Yang, Yang Gao, Hui-Dong Liu, and Yilong Yin. 2019. Joint multi-label classification and label correlations with missing labels and feature selection. *Knowledge-Based Systems* 163 (2019), 145–158.

[17] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. 2008. Label ranking by learning pairwise preferences. *Artificial Intelligence* 172, 16-17 (2008), 1897–1916.

[18] Muhammad Iqbal, Will N Browne, and Mengjie Zhang. 2013. Evolving optimum populations with XCS classifier systems. *Soft Computing* 17, 3 (2013), 503–518.

[19] Tobias Jordan, Philippe de Wilde, and Fernando Buarque de Lima Neto. 2020. Decision making for two learning agents acting like human agents: A proof of concept for the application of a Learning Classifier Systems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[20] Ji-Yoon Kim and Sung-Bae Cho. 2019. Exploiting deep convolutional neural networks for a neural-based learning classifier system. *Neurocomputing* 354 (2019), 61–70.

[21] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. 2012. An extensive experimental comparison of methods for multi-label learning. *Pattern recognition* 45, 9 (2012), 3084–3104.

[22] Shabnam Nazmi, Xuyang Yan, and Abdollah Homaifar. 2018. Multi-label Classification Using Genetic-Based Machine Learning. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 675–680.

[23] Shabnam Nazmi, Xuyang Yan, Abdollah Homaifar, and Emily Doucette. 2020. Evolving multi-label classification rules by exploiting high-order label correlations. *Neurocomputing* 417 (2020), 176–186.

[24] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2011. Classifier chains for multi-label classification. *Machine learning* 85, 3 (2011), 333.

[25] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer science review* 1, 1 (2007), 27–64.

[26] Robert E Schapire and Yoram Singer. 2000. BoosTexter: A boosting-based system for text categorization. *Machine learning* 39, 2-3 (2000), 135–168.

[27] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[28] Cees GM Snoek, Marcel Worring, Jan C Van Gemert, Jan-Mark Geusebroek, and Arnold WM Smeulders. 2006. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM international conference on Multimedia*. 421–430.

[29] Jinhui Tang, Richang Hong, Shuicheng Yan, Tat-Seng Chua, Guo-Jun Qi, and Ramesh Jain. 2011. Image annotation by k nn-sparse graph-based label propagation over noisily tagged web images. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 2 (2011), 1–15.

[30] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2008. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, Vol. 21. sn, 53–59.

[31] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. 2010. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering* 23, 7 (2010), 1079–1089.

[32] Grigorios Tsoumakas and Ioannis Vlahavas. 2007. Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*. Springer, 406–417.

[33] Ryan Urbanowicz, Ambrose Granizo-Mackenzie, and Jason Moore. 2012. Instance-linked attribute tracking and feedback for michigan-style supervised learning classifier systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 927–934.

[34] Ryan Urbanowicz, Niranjan Ramanand, and Jason Moore. 2015. Continuous endpoint data mining with exstracs: A supervised learning classifier system. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1029–1036.

[35] Ryan John Urbanowicz, Angeline S Andrew, Margaret Rita Karagas, and Jason H Moore. 2013. Role of genetic heterogeneity and epistasis in bladder cancer susceptibility and outcome: a learning classifier system approach. *Journal of the American Medical Informatics Association* 20, 4 (2013), 603–612.

[36] Ryan J Urbanowicz and Will N Browne. 2017. *Introduction to learning classifier systems*. Springer.

[37] Ryan J Urbanowicz, Delaney Granizo-Mackenzie, and Jason H Moore. 2012. Using expert knowledge to guide covering and mutation in a michigan style learning classifier system to detect epistasis and heterogeneity. In *International Conference on Parallel Problem Solving from Nature*. Springer, 266–275.

[38] Ryan J Urbanowicz and Moshe Sipper. 2020. Evolutionary algorithms in biomedical data mining: challenges, solutions, and frontiers. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1224–1253.

[39] Rosane MM Vallim, David E Goldberg, Xavier Llorà, Thyago SPC Duque, and André CPLF Carvalho. 2008. A new approach for multi-label classification based on default hierarchies and organizational learning. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. 2017–2022.

[40] Stewart W Wilson. 1995. Classifier fitness based on accuracy. *Evolutionary computation* 3, 2 (1995), 149–175.

[41] Stewart W Wilson. 2001. Function Approximation with a Classi er System. In *Proc. 3rd Genetic and Evolutionary Computation Conf.(GECCO'01)(p. 974Â981)*. Citeseer.

[42] Stewart W Wilson. 2002. Classifiers that approximate functions. *Natural Computing* 1, 2 (2002), 211–234.

[43] Min-Ling Zhang and Zhi-Hua Zhou. 2007. ML-KNN: A lazy learning approach to multi-label learning. *Pattern recognition* 40, 7 (2007), 2038–2048.

[44] Min-Ling Zhang and Zhi-Hua Zhou. 2013. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering* 26, 8 (2013), 1819–1837.