Multi-label classification search space in the MEKA software

Alex G. C. de Sá[1], Alex A. Freitas[2], and Gisele L. Pappa[1]

[1] Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brazil
[2] School of Computing, University of Kent at Canterbury (UKC), UK
alexgcsa@dcc.ufmg.br, A.A.Freitas@kent.ac.uk, glpappa@dcc.ufmg.br

Abstract. This technical report describes the multi-label classification (MLC) search space in the MEKA software, including the traditional/meta MLC algorithms, and the traditional/meta/preprocessing single-label classification (SLC) algorithms. The SLC search space is also studied because is part of MLC search space as several methods use problem transformation methods to create a solution (i.e., a classifier) for a MLC problem. This was done in order to understand better the MLC algorithms. Finally, we propose a grammar that formally expresses this understatement.

Keywords: Multi-Label Classification, Search Space, Grammar.

1 Search Space – Algorithms from WEKA

In this section, we study **22** traditional (single label) classification algorithms from the WEKA software [27]. This is done in order to understand the whole search space of multi-label methods. All parameters in this section were set in accordance to the search space definition from Auto-WEKA [62,40,43]. The methods and their respective (hyper-)parameters were defined after studing the code, logs and configuration files of Auto-WEKA, which is considered a stable and robust approach for automatically selecting and configuring machine learning algorithms.

1.1 C4.5 (J48) [48] – Trees

The method for generating a C4.5 decision tree. This algorithm can decide whether it will use the default C4.5's error-based pruning method [7,59,68] or not. If the algorithm decides to use pruning, the C4.5's pruning method is applied to the tree, and an estimation of the error rate of every subtree is done. After that, the pruning method will replace the subtree with a leaf node if the estimated error of the leaf is lower than a threshold [59]. **Parameters:**

– Confidence factor (cf)[-C]: It is used for C4.5's error-based pruning method (smaller values incur more pruning) and is defined by the interval: $\{cf \in \mathbb{R} \mid 0.0 \leq cf \leq 1.0\}$.

– Minimum number of objects (mno)[-M]: The minimum number of instances per leaf. It can take values in the interval:
$\{mno \in \mathbb{Z} \mid 1 \leq mno \leq 64\}$.
– Collapse tree (ct)[-O]: It is used to decide if internal nodes will be collapsed to avoid overfitting. This parameter is used with C4.5's error-based pruning method to enhance the final decision tree. It collapses a subtree to a node only if training error of the subtree does not increase when compared to the entire tree. It is applied to every subtree in the tree, where subtrees are collapsed (pruned) if pruning does not increase its classification error. For example, if there is a subtree with two leaf nodes having the same classification on the training data, this subtree will be replaced by a single leaf. It can take boolean values (true or false).
– Unpruned (u)[-U]: It decides whether pruning is performed or not. It can take boolean values (true or false).
– Binary splits (bs)[-B]: It decides whether C4.5 will use binary splits on nominal attributes when building the trees. It can take boolean values (true or false).
– Use MDL correction (umc)[-J]: It decides whether the MDL correction is used when finding splits on numeric attributes. It can take boolean values (true or false).
– Use Laplace (ul)[-A]: It decides if the counts of instances at leaves are smoothed based on the Laplace correction. It can take boolean values (true or false).
– Subtree raising (sr)[-S]: It is used for C4.5's error-based pruning and decides whether the algorithm will consider the subtree raising operation when pruning. It can take boolean values (true or false).

Dependencies/Constraints:

1. If the parameter unpruned is set to "true", the parameters "confidence factor", "collapse tree" and "subset raising" are not used (omitted).

1.2 Logistic model trees (LMT) [42,61] – Trees

The method for building logistic model trees (LMT), which are classification trees with logistic regression functions at the leaves. This is done by using the LogitBoost algorithm. In this case, boosting is used (aiming) to build very effective decision trees. The idea of LMT is to use LogitBoost to induce trees with linear-logistic regression models at the leaves. LogitBoost performs additive logistic regression. Thus, at each iteration of the boosting algorithm, it creates a simple regression model by going through all the attributes, finding the simple regression function with the smallest error, and adding it into the additive model [68]. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes and missing values. **Parameters:**

– Convert Nominal (cn)[-B]: It decides if the method will convert all nominal attributes to binary ones before building the tree. This means that all splits in the final tree will be binary. It can take boolean values (true or false).

- Split on residuals (*sor*)[-R]: It decides whether the method will set the splitting criterion based on the residuals of LogitBoost. There are two possible splitting criteria for LMT: the default is to use the C4.5 splitting criterion that uses information gain on the class variable. The other splitting criterion tries to improve the purity in the residuals produced when fitting the logistic regression functions. It can take boolean values (true or false).
- Fast Regression (*fr*)[-C]: It decides whether the method will use a heuristic that avoids the use of cross-validation to optimize the number of LogitBoost iterations at every node. In the case of using this heuristic, LMT will fit the logistic regression functions at a leaf node using the LogitBoost algorithm, applying a 5-fold cross-validation procedure to determine how many iterations to run just once. Then, it employs the same number of iterations throughout the tree, instead of cross-validating at every node. This heuristic reduces the running time considerably, with little effect on accuracy [68]. It can take boolean values (true or false).
- Error on probabilities (*eop*)[-P]: It decides if the method will minimize the error on classification probabilities instead of the misclassification error when cross-validating the number of LogitBoost iterations. When this parameter is set to 'true', the number of LogitBoost iterations that minimizes the error on classification probabilities instead of the misclassification error is chosen. It can take boolean values (true or false).
- Weight trim beta(*wtb*)[-W]: It sets the beta value used for weight trimming in LogitBoost. Only instances carrying (1 - beta)% of the weight from the previous iteration are used in the next iteration. The value zero (0) means no weight trimming, which is the default value. The values are restricted to the interval : $\{wtb \in \mathbb{R} \mid 0.0 \leq wtb \leq 1.0\}$. It can also be omitted and take the default value of zero with 50% of probability. The other values jointly take the other 50% of probability.
- Use AIC (*uaic*)[-A]: It decides if the method will use the AIC (Akaike's Information Criterion) measure to determine when to stop LogitBoost's iterative process. More precisely, if *uaic* takes the value 'true', the best number of iterations will be defined by an information criterion measure (currently, AIC). If false, the stopping criterion will be determined by the best number of iterations in a 5-fold cross-validation procedure. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of LMT.

1.3 Decision Stump (DS) [68] – Trees

The method for building and applying a decision stump model, which is considered a weak learner. Because of that, it is usually used in conjunction with a boosting algorithm.

The DS's classification is based on the entropy measure and a missing value is treated as a separate value. The DS algorithm constructs a simple decision tree that has only one level, i.e., a decision tree that has only one internal (root)

node, that is directly linked to the leaves. It also creates an extra branch for missing values.

In the case of nominal attributes at the root node, there are two possibilities. The first possibility is to build a stump which contains a leaf for each possible feature value. The second possibility is to consider a stump with two leaves, one of them is mapped to some category, and the another to all other categories. The DS from WEKA employs the latter approach. This method has no explicit parameters.

1.4 Random Forest (RF) [9] – Trees

The method for constructing a forest of random trees. **Parameters:**

- Number of trees (nt)[-I]: The number of trees to be generated by the algorithm. It is an integer value bounded by the interval: $\{nt \in \mathbb{Z} \mid 2 \leq nt \leq 256\}$.
- Number of features (nf)[-K]: It sets the number of randomly sampled attributes used as candidate attributes at each tree node. It is an integer value bounded by the interval: $\{nf \in \mathbb{Z} \mid 2 \leq nf \leq 32\}$. However, it may also take the value zero (0) with 50% of probability, which means nf will be just used as a flag to indicate that the real value produced by the equation $log_2(number\_of\_attributes + 1)$ rounded to the nearest integer is automatically used for this parameter. The values from two (2) to 32 jointly take the other 50% of probability.
- Maximum depth (md)[-depth]: The maximum depth of the tree. It is bounded by the interval: $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$. However, it may also take the value zero (0) as a flag with 50% of probability and, in this case, the depth of the tree can be unlimited. The values from two (2) to 20 jointly take the other 50% of probability.

There are no dependencies/constraints between the parameters of RF.

1.5 Random Tree (RT) [68] – Trees

The method for constructing a tree that considers K randomly sampled attributes as candidate attributes at each node. It is important to mention that this version of RT performs no pruning. **Parameters:**

- Minimum weight (mw)[-M]: The minimum total weight of the instances in a leaf. It is restricted by the interval:
  $\{mw \in \mathbb{Z} \mid 1 \leq mw \leq 64\}$.
- Number of features (nf)[-K]: It sets the number of randomly sampled attributes used as candidate attributes at each tree node. It is bounded by the interval: $\{nf \in \mathbb{Z} \mid 2 \leq nf \leq 32\}$. However, it may also take the value zero (0) with 50% of probability, which means nf will be just used as a flag to indicate that the real value produced by the equation $log_2(number\_of\_attributes+1)$ rounded to the nearest integer is automatically used for this parameter. The values from two (2) to 32 jointly take the other 50% of probability.

– Maximum depth (md)[-depth]: The maximum depth of the tree. It is bounded by the interval: $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$. However, it may also take the value zero (0) as a flag with 50% of probability and, in this case, the depth of the tree is unlimited. The values from two (2) to 20 jointly take the other 50% of probability.
– Number of folds for back-fitting and for growing the tree ($nfbgt$)[-N]: It determines the amount of data used for back-fitting and for growing the tree. One fold is used for back-fitting, i.e., for making a preliminary estimation of class probabilities based on a hold-out set. The others (nf - 1) folds are used for growing the tree. It is bounded by the interval: $\{nfbf \in \mathbb{Z} \mid 2 \leq nfbf \leq 5\}$. It can also use the value zero (0) with 50% of probability, which means no back-fitting will be performed in this case. Thus, the values from two (2) to five (5) jointly take the other 50% of probability. It can not take the value one (1) because we would have zero folds for growing the tree. In the case of taking the value one, the algorithm returns an error and does not run. It is important to mention that Auto-WEKA allows this error, ignoring RT algorithm with this configuration (when it occurs), and continuing the search from this point.

There is no constraints/dependencies between the parameters of RT.

1.6 REPTree [68] – Trees

The method for the fast decision tree learner. It builds a decision tree using information gain and prunes it using reduced-error pruning (with back-fitting). It only sorts values for numeric attributes once, at the start of the algorithm. Missing values are dealt with by splitting the corresponding instances into pieces (i.e., as in C4.5).

Parameters:

– Minimum weight (mw)[-M]: The minimum total weight of the instances in a leaf. It is restricted by the interval:
$\{mw \in \mathbb{Z} \mid 1 \leq mw \leq 64\}$.
– Maximum depth (md)[-L]: The maximum tree depth. It can take integer values considering the interval: $\{md \in \mathbb{Z} \mid 2 \leq md \leq 20\}$. However, it may also take the value -1 as a flag with 50% of probability and, in this case, the depth of the tree the depth will not be restricted. The values from two (2) to 20 jointly take the other 50% of probability.
– Use pruning (up)[-P]: It decides whether REPTree will use reduced-error pruning or not. In the case of using this pruning method, a simple hold-out set ($\frac{1}{3}$ of the training data) is used to estimate the error of a node, instead of using cross-validation. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of REPTree.

1.7 Decision Table (DT) [39] – Rules

The method for building and using a simple decision table classifier. **Parameters:**

– *Evaluation Measure* (em)[-E]: The measure used to evaluate the performance of attribute combinations used in the decision table. It can take one of the four categorical values: 1. accuracy (acc); 2. root mean squared error (rmse) of the the class probabilities; 3. mean absolute error (mae) of the class probabilities; 4. area under the ROC curve (auc). The two measures *rmse* and *mae* are adapted to be used in the classification context.
– Use IBk (*uibk*)[-I]: It sets whether a k-nearest neighbor (k=1) classifier should be used instead of the majority class in order to classify non-matching instances. It can take boolean values (true or false).
– Search method (*sm*)[-S]: It sets the search method which will be used to find good attribute combinations for the decision table. It can take the values GreedyStepwise or BestFirst.
– Cross-Validation (*crv*)[-X]: It sets the number of folds for the internal cross validation procedure to evaluate the attribute sets. It may take the values one (1), two (2), three (3) or four (4). If the value 1 is set for this parameter, a leave one out procedure is applied.

There are no dependencies/constraints between the parameters of DT.

1.8 JRip [12] – Rules

The method that implements a propositional rule learner algorithm, namely Repeated Incremental Pruning to Produce Error Reduction (RIPPER). **Parameters:**

– Minimum total weight (*mtw*)[-N]: This parameter determines the minimum total weight of the instances in a rule. It can take values considering the interval: $\{mtw \in \mathbb{R} \mid 1.0 \leq mtw \leq 5.0\}$.
– Check error rate (*cer*)[-E]: It decides whether JRip will consider the "error rate greater or equal than 0.5" as a stopping criterion. It can take boolean values (true or false).
– Use pruning (*up*)[-P]: It decides whether JRip will use reduced error pruning or not. In the case of using this pruning method, a 3-fold cross-validation procedure is applied to prune the rules. Otherwise, no pruning method is used. It can take boolean values (true or false).
– Optimizations (*o*)[-O]: The number of optimization runs. It can take integer values considering the interval: $\{o \in \mathbb{Z} \mid 1 \leq o \leq 5\}$.

There are no dependencies/constraints between the parameters of JRip.

1.9 OneR [33] – Rules

The method for building and using a 1R classifier. In other words, it uses the minimum-error attribute for prediction, discretizing numeric attributes. **Parameters:**

– Minimum bucket size (*mbs*)[-B]: It is used for discretizing numeric attributes. It is limited by the interval: $\{mbz \in \mathbb{Z} \mid 1 \leq mbz \leq 32\}$.

OneR has only one parameter and, consequently, there is no dependencies/constraints for it.

1.10 PART [21] – Rules

The method for generating a PART decision list. PART uses the separate-and-conquer paradigm: It builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule. **Parameters:**

- Minimum number of objects (mno)[-M]: The minimum number of instances per leaf. It can take values in the interval:
  $\{mno \in \mathbb{Z} \mid 1 \leq mno \leq 64\}$.
- Binary splits (bs)[-B]: It decides whether C4.5 will use binary splits on nominal attributes when building the trees. It can take boolean values (true or false).
- Reduced-error pruning(rep)[-R]: It is used to decide whether reduced-error pruning is used instead of C4.5's default pruning (error-based pruning). If C4.5's error-based pruning is chosen, a (default) confidence factor of 0.25 is used to prune the tree. If not (i.e, the reduced-error pruning is chosen), the method will consider each node for pruning and the removal of a subtree at a node is done if the resulting tree performs no worse than the original one on the validation set. The size of the validation set is determined by the next parameter (nr). It can take boolean values (true or false).
- Number of folds (nr)[-N]: It determines the amount of data used for reduced-error pruning. One fold is used for pruning and the rest for growing the tree. It can take the values two (2), three (3), four (4) or five (5).

Dependencies/Constraints:

1. If the reduced-error pruning method is not set to "true", the parameter "number of folds" is not used.

1.11 ZeroR [68] – Rules

The method for building and using a 0-R classifier. The ZeroR classifier simply predicts the majority category (class), ignoring the predictor attributes. This method has no explicit parameters.

1.12 K-Nearest Neighbors (KNN) [3] – Lazy

The method for K-Nearest Neighbors classifier. KNN can select an appropriate value of K based on internal leave-one-out evaluation and can also compute distances based on instance weighting. **Parameters:**

- Number of neighbors (k)[-K]: The number of neighbors to use. The value of k is bounded by the interval: $\{k \in \mathbb{Z} \mid 1 \leq k \leq 64\}$.
- Leave-one-out (loo)[-X]: It decides whether leave-one-out evaluation on the training data will be used or not to select the best k value between 1 and the value specified as the KNN parameter. If set as false, the selected k value is used. It can take only boolean values (true or false).

– Distance weighting (dw): It sets the used distance weighting method. It may take the unique following values:
  - -I: Weight neighbors by the inverse of their distance.
  - -F: Weight neighbors by one minus their distance.
  - None: No distance weighting method is applied.

There are no dependencies/constraints between the parameters of KNN.

1.13 K* [11] – Lazy

The method K* is an instance-based classification algorithm. Thus, in order to classify a test instance, K* considers the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners by using an entropy-based distance function. **Parameters:**

– Global blending (gb)[-B]: The parameter is a percentage for global blending. This parameter controls the "sphere of influence" by specifying how many of the neighbors of the instance i should be considered important (although there is no hard cut off at the edge of the sphere – it is more related to a gradual decreasing of importance). The values are restricted to the interval $\{gb \in \mathbb{Z} \mid 1 \leq gb \leq 100\}$. Thus, selecting zero (0) for this parameter gives a nearest neighbor algorithm (this is why Auto-WEKA does not allow to choose it), and choosing 100 gives equally weighted instances. Intermediate values are interpolated linearly.
– Entropic auto-blending (eab)[-E]: It decides whether entropy-based blending will be used or not. It can take boolean values (true or false).
– Missing Mode (mm)[-M]: It determines how missing attribute values are treated. It can take one of the four categorical values: 1. average column entropy curves (a); 2. ignore the instances with missing values (d); 3. treat missing values as maximally different (m); 4. normalize over the attributes (n).

There are no dependencies/constraints between the parameters of K*.

1.14 Voted Perceptron (VP) [23] – Functions

The voted perceptron algorithm created by Freund and Schapire. It globally replaces all missing values by their default values. More precisely, VP replaces all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. Additionally, it transforms nominal attributes into binary ones. **Parameters:**

– Number of iterations (i)[-I]: The number of iterations to be performed by VP. This parameter varies in accordance to the interval:
$\{i \in \mathbb{Z} \mid 1 \leq i \leq 10\}$.
– Max K(mk)[-M]: The maximum number of alterations to the perceptron, i.e., the maximum number of perceptrons used in the iterative process. It can take values of the interval:
$\{mk \in \mathbb{Z} \mid 5,000 \leq mk \leq 50,000\}$

– Exponent (e)[-E]: The exponent for the polynomial kernel. It can take values of the interval: $\{e \in \mathbb{R} \mid 0.2 \leq e \leq 5.0\}$

There are no dependencies/constraints between the parameters of VP.

1.15 Multilayer Perceptron (MLP) [58] – Functions

A method that uses the back-propagation algorithm to classify instances. MLP creates just one hidden layer (for now) and all its nodes use sigmoid activation functions (except for when the class is numeric in which case the the output nodes become unthresholded linear units). **Parameters:**

– Learning rate (lr)[-L]: The amount by which the weights are updated during training. It is restricted by the interval:
$\{lr \in \mathbb{R} \mid 0.1 \leq lr \leq 1.0\}$.
– Momentum (m)[-M]: It is applied to the weights during updating. It is restricted by the interval: $\{m \in \mathbb{R} \mid 0.0 \leq m \leq 1.0\}$.
– Number of hidden nodes (nhn)[-H]: It defines the number of hidden nodes in the hidden layer of the neural network. This parameter may take four predefined nominal values (a, i, o and t), which represent the following integer values:
  • a = $\frac{(number\_of\_attributes + number\_of\_classes)}{2}$, always using the default floor function to convert it to an integer value.
  • i = number of attributes.
  • o = number of classes.
  • t = $(number\_of\_attributes + number\_of\_classes)$.
– Nominal to binary filter ($n2b$)[-B]: It decides whether the algorithm will transform nominal attributes to binary ones or not. This could help improve performance if there are nominal attributes in the data. It can take boolean values (true or false).
– Reset (r)[-R]: It decides whether the algorithm will use the reset approach. In this case, the algorithm will allow the network to reset with a lower learning rate. If the network diverges from the answer, this will automatically reset the network with a lower learning rate and begin training again. It can take boolean values (true or false).
– Decay (d)[-D]: It decides whether the algorithm will cause the learning rate to decrease. This will divide the starting value of the learning rate by the sequential number of the current epoch in order to determine what the current learning rate should be. This may help to stop the network from diverging from the target output, as well as improving general performance. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of MLP.

1.16 Stochastic Gradient Descent (SGD) [68] – Functions

The method that implements the stochastic gradient descent approach for learning various linear models (binary class SVM, binary class logistic regression,

squared loss, Huber loss and epsilon-insensitive loss linear regression). **Parameters:**

- Loss function (lf)[-F]: It sets the loss function to be minimized. It can take the following integer values associated to three approaches:
  - (0): hinge loss (SVM).
  - (1): log loss (logistic regression).
  - (2): squared loss (regression).
- Learning rate (lr)[-L]: The learning rate. If normalization is turned off, then the default learning rate will need to be reduced. It is restricted by the interval: $\{lr \in \mathbb{R} \mid 0.00001 \leq lr \leq 1.0\}$.
- Ridge (r)[-R]: It sets the Ridge value in the log-likelihood. This parameter can take any value of the given set:
  $\{r \in \mathbb{R} \mid 10^{-12} \leq r \leq 10.0\}$
- Do not normalize (nn)[-N]: It decides whether normalization will be turned off or not. It can take boolean values (true or false).
- Do not replace missing values ($nrmv$)[-M]: It decides whether global replacement of missing values will be turned off or not. In the case of being turned off, the missing values will be ignored. Otherwise, SGD will replace all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of SGD.

1.17 Sequential Minimal Optimization (SMO) [47,36,28] – Functions

This method implements John Platt's sequential minimal optimization algorithm for training a support vector classifier (SVC). It globally replaces all missing values by their default values. More precisely, SMO (like VP) replaces all missing values for nominal and numeric attributes by the modes and the means from the training data, respectively. Additionally, it transforms nominal attributes into binary ones.

Parameters:

- Cost (c)[-C]: It defines the complexity parameter, which is the penalty parameter of the error term and is defined by the interval: $\{c \in \mathbb{R} \mid 0.5 \leq c \leq 1.5\}$. This is a parameter that controls the trade-off between training error and model complexity. It is important to mention that a low value of c will increase the number of training errors, whereas a high value of c will lead to a behavior similar to that of a hard-margin SVM [34].
- Filter type (ft)[-N]: It determines how/if the data will be transformed. It may take the values zero (0, i.e, normalize the training data – it sets all the numeric attributes in the given dataset into the interval [0,1]), one (1, i.e, standardize the training data – it standardizes all numeric attributes in the given dataset to have zero mean and unit variance) or two (2, i.e. no normalization/standardization is applied to the data).

– Build Calibration Models (bcm)[-M]: It decides whether the model will fit calibration models to SVM's outputs (for proper probability estimates). It can take boolean values (true or false).
– Kernel(k)[-K]: The kernel to use. It can take one of the following possible kernels (and associated constrained parameters):
  • NormalizedPolyKernel: The normalized polynomial kernel. **Parameters:**
    1. Exponent (exp)[-E]: It determines the exponent value and is defined by the interval: $\{exp \in \mathbb{R} \mid 0.2 \leq exp \leq 5.0\}$.
    2. Use Lower-Order (ulo)[-L]: It decides whether the method will use lower-order terms or not. It can take boolean values (true or false).
  • PolyKernel: The standard polynomial kernel. **Parameters:**
    1. Exponent (exp)[-E]: It determines the exponent value and is defined by the interval: $\{exp \in \mathbb{R} \mid 0.2 \leq exp \leq 5.0\}$.
    2. Use Lower-Order (ulo)[-L]: It decides whether the method will use lower-order terms or not. It can take boolean values (true or false).
  • Puk: The Pearson VII function-based universal kernel [66]. **Parameters:**
    1. Omega (om)[-O]: The omega value. It is defined by the interval: $\{om \in \mathbb{R} \mid 0.1 \leq om \leq 1.0\}$.
    2. Sigma (sig)[-S]: The sigma value. It is defined by the interval: $\{sig \in \mathbb{R} \mid 0.1 \leq sig \leq 10.0\}$.
  • RBF:The RBF kernel. **Parameters:**
    1. Gamma (g)[-G]: The gamma value. It is defined by the interval: $\{g \in \mathbb{R} \mid 0.0001 \leq g \leq 1.0\}$.

The constraints/dependencies for SMO are only in the selection of the kernel and its respective parameters.

1.18 Logistic Regression (LogR) [10] – Functions

Method for building and using a multinomial logistic regression model with a ridge estimator. **Parameters:**

– Ridge (r)[-R]: It sets the Ridge value in the log-likelihood. This parameter can take any value of the given set:
$\{r \in \mathbb{R} \mid 10^{-12} \leq r \leq 10.0\}$

LogR has one parameter and, consequently, there is no dependencies/constraints for it.

1.19 Simple Logistic (SL) [42,61] – Functions

The method for constructing logistic regression models. LogitBoost with simple regression functions as base learners is used for fitting the logistic models. **Parameters:**

– Weight trim beta (*wtb*)[-W]: It sets the beta value used for weight trimming in LogitBoost. Only instances carrying (1 - beta)% of the weight from the previous iteration are used in the next iteration. The value zero (0) means no weight trimming, which is the default value. The values are restricted to the interval : $\{wtb \in \mathbb{R} \mid 0.0 \leq wtb \leq 1.0\}$. It also can be omitted and take the default value of zero. This is a constraint that should be defined into the grammar.
– Use Cross-Validation (*ucv*)[-S]: It decides if SL will try to find the best number of LogitBoost iterations using an internal 5-fold cross-validation procedure or simply using the number of iterations that minimizes error on the training set. Thus, if not set to 'true', the number of LogitBoost iterations which is used is the one that minimizes the error on the training set (misclassification error). It can take boolean values (true or false).
– Use AIC (*uaic*)[-A]: It decides if the method will use the AIC (Akaike's Information Criterion) measure to determine when to stop the LogitBoost iterative process. More precisely, if *uaic* takes the value 'true', the best number of iterations will be defined by an information criterion measure (currently, AIC). If false, the stopping criterion will be determined by the best number of iterations in an internal 5-fold cross-validation procedure or simply in accordance to the error on the training set, as explained in the previous item. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of SL.

1.20 Naïve Bayes (NB) [35] – Bayes

The Naïve Bayes classifier using estimator classes. This algorithm builds a fixed structure (model) given the attributes of the dataset. **Parameters:**

– Use kernel estimator (*uke*)[-K]: It decides whether NB will use a kernel estimator for numeric attributes rather than a (single) Gaussian distribution. In the case of using the kernel estimator, NB will apply one Gaussian kernel per observed data value (for more details, see Flexible Naïve Bayes' section in [35]). It can take boolean values (true or false). It is important to mention that a discrete estimator is automatically used for nominal attributes, which is a simple discrete probability estimator based on nominal values' counts. This also means the Laplace correction is applied in order to perform the estimation.
– Use supervised distribution (*usd*)[-D]: It decides whether NB will use supervised discretization to convert numeric attributes to nominal ones. Discretization is performed by Fayyad and Irani's method [18]. This method uses a criterion based on the minimum description length (MDL) principle to define the number of intervals produced over the continuous space [16]. It can take boolean values (true or false).

Dependencies/Constraints:

1. If the parameter "use kernel estimator" is activated, the parameter "use supervised distribution" must not be activated; and vice-versa. This constraint must be enforced in the grammar.

1.21 Bayesian Network Classification (BNC) Algorithms [6] – Bayes

This method is used to learn a Bayesian Network Classifier based on various search algorithms and a local Bayesian scoring metric [13,29]. **Parameters:**

– Search Method (*sm*)[-Q]: For BNC algorithms, the optimization occurs just on the method used for searching network structures. Thus, the search method can be one of the following: 1. Tree Augmented Naïve Bayes (TAN) [24]; 2. K2 [13]; 3. Hill Climbing (HC) [5,30]; 4. Look Ahead in Good Directions Hill Climbing (LAGDHC) [1]; 5. Simulated Annealing (SA) [5]; 6. Tabu Search (TS) [5]. All the search methods uses the parameter "maximum number of parents" set to two (including the class node), except for TAN and SA which do not use the "maximum number of parents" as a parameter. In addition, the methods use the (default) Bayesian scoring metric to search for appropriate Bayesian networks to data.

BNC has one parameter and, consequently, there is no dependencies/constraints for it.

1.22 Naïve Bayes Multinomial (NBM) [2,19,44,46,68] – Bayes

The method for building and using a multinomial version of Naïve Bayes. This algorithm was particularly designed for text classification and, for this reason, it changes how the traditional Naïve Bayes calculates the probabilities. This is done to take into account the number of times a word appears in the document.

This method has no explicit parameters and, consequently, there is no dependencies/constraints for it.

2 Search Space – Meta classification algorithms from WEKA

In this section, the search space of **7** traditional (single label) meta classification algorithms from WEKA [27] is studied. This is also done in order to extend and improve the search space of multi-label methods. All parameters in this section were also set in accordance to the search space definition from Auto-WEKA [62,40,43]. The methods and their respective (hyper-)parameters were defined after studying the code, logs and configuration files of Auto-WEKA, which is considered a stable and robust approach for automatically selecting and configuring machine learning algorithms.

2.1 Locally Weighted Learning (LWL) [20,4]

The locally weighted learning method. It uses an instance-based algorithm to assign instance weights which are then used by a specified Weighted Instances Handler. In other words, LWL assigns weights using an instance-based method and, after this step, another classification algorithm is used to build a classifier from the weighted instances. For example, it can do the classification by using a naïve Bayes classifier or a decision stump (default) from these weighted instances.
Parameters:

- Classifier (c)[-W]: The classifier to be used. It can be one classification algorithm from Section 1, except for the algorithms LMT, OneR, K*, SGD and VT, as these the classifiers produced by these algorithms do not handle weighted instances.
- Number of neighbors (k)[-K]: It sets how many neighbors are used to determine the width of the weighting function. It may take the following values: $\{-1, 10, 30, 60, 90, 120\}$. A negative value means that all neighbors will be considered. Additionally, it can be omitted with 50% of probability, taking automatically the negative value -1. The six (6) not omitted values jointly take the other 50% of probability, which represents at the end that the value -1 has 58.33% of probability to be chosen. This constraint must be considered into the grammar.
- Weighting kernel (wk)[-U]: It determines the weighting function and may take the five following integer values:
  - (0) Linear.
  - (1) Epnechnikov.
  - (2) Tricube.
  - (3) Inverse.
  - (4) Gaussian.

  It can be omitted with 50% of probability, and then LWL will use the default value zero for this parameter, i.e., the linear function. The five (5) not omitted values jointly take the other 50% of probability, which represents at the end that the value 0 has 60% of probability to be chosen.

There are no dependencies/constraints between the parameters of LWL.

2.2 Random Subspace (RSS) [32]

This method constructs an ensemble classifier that consists of multiple models systematically constructed by randomly selecting subsets of components of the feature vector, i.e., the classification models are constructed according to random subspaces. More precisely, for each classifier, a certain percentage of the number of attributes is randomly sampled and then used to build the classifier. **Parameters:**

- Classifier (c)[-W]: The classifier to be used. It can be any classification algorithm from Section 1.
- Subspace size (sss)[-P]: It defines the size of each sub-space as a percentage of the number of attributes. It could take values in the range: $\{sss \in \mathbb{R} \mid 0.1 \leq sss \leq 1.0\}$.
- Number of iterations (ni)[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range:
  $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 64\}$.

  There are no dependencies/constraints between the parameters of RSS.

2.3 Bagging [8]

The method for bagging a classifier in order to reduce variance. **Parameters:**

- Classifier (c)[-W]: The classifier to be used for each member of the ensemble. It can be any classification algorithm from Section 1.
- Bag size percent (bsp)[-P]: It defines the size of each bag, as a percentage of the training set size. It may take values in the range: $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$. It makes sampling with replacement. Thus, even if the bag size percent is 100%, it will sample different sets with the same size of the training set.
- Number of iterations (ni)[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range:
  $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 128\}$.
- Calculate out-of-bag ($coob$)[-O]: It decides whether the out-of-bag error is calculated. It can take boolean values (true or false).

  Dependencies/Constraints:

1. If the parameter "calculate out-of-bag" is activated (set to true), the parameter "bag size percent" must be equal to 100. This is a constraint of WEKA and only an internal modification in the WEKA code could suppress it. This can happen with 50% of probability. I.e., in half of the cases, the parameter "bag size percent" is set to 100. In the other part of the cases, "bag size percent" may take values between 10 and 100, because the parameter "calculate out-of-bag" is not activated (set to false).

2.4 Random Committee (RC) [68]

The method for building an ensemble of randomizable base classifiers from WEKA. For this reason, the only classifiers (at the base level) that can be used in for this meta-algorithm are Random Forest (RF), Random Tree (RT), REP Tree (REPTree), Stochastic Gradient Descent (SGD) and Multilayer Perceptron (MLP). The creation of a (pseudo-)randomizable classifier is done by using an input seed. It is important to mention that the classifiers in the ensemble differ in terms the structure of their models. For instance, a random seed can define how the random trees are constructed in RF, RT and REPTree, how the linear models are defined in SGD, and how the network connection weights are firstly defined in MLP. Nevertheless, all classifiers are constructed using the same data, differently from Bagging and RSS. Thus, at the end, the final prediction is based on the average of the class probabilities generated by the base classifiers. **Parameters:**

- Classifier (c)[-W]: The classifier to be used for each member of the ensemble. It is restricted in one of the five (5) algorithms aforementioned, i.e., RF, RT, REPTree, SGD and MLP.
- Number of iterations (ni)[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take values in the range: $\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 64\}$.

There are no dependencies/constraints between the parameters of RC.

2.5 Ada Boost M1 (AdaM1) [22]

The method for boosting a nominal class classifier using the Adaboost M1 approach. **Parameters:**

- Classifier (c)[-W]: The classifier to be used. It can be one classification algorithm from Section 1, except for the algorithms LMT, OneR, K*, SGD and VT, as these the classifiers produced by these algorithms do not handle weighted instances. It is important to mention than Auto-WEKA allows any classifier at the base level of AdaM1, including those which can not handle weights in the instances. In this case, Auto-WEKA ignores that algorithm (with its configuration) and proceeds with the search.
- Weight threshold (wt)[-P]: It defines the weight threshold for weighted pruning, i.e., it only selects instances with weights that contribute to the specified quantile of the weight distribution. It may take values in the range: $\{wt \in \mathbb{Z} \mid 10 \leq wt < 100\}$.
  However, it may also automatically take the value 100, as a specific flag, disregarding the possible values of the range. In this case, there is no instance pruning as all the instances are considered to build the model. This constraint happens with 50% of probability and must be considered into the grammar.

– Number of iterations (ni)[-I]: It defines the number of iterations to be performed, i.e., the number of classifiers in the ensemble. It may take the values in the range:
$\{ni \in \mathbb{Z} \mid 2 \leq ni \leq 128\}$.
– Use resampling (ur)[-Q]: It decides whether AdaM1 will use resampling instead of reweighting. Thus, it is possible to generate an unweighted dataset from the weighted data by resampling. In this case, instances are chosen with probability proportional to their weight. As a result, instances with high weight are replicated frequently, and the ones with low weight may never be selected. Once the new dataset becomes as large as the original one, it is fed into the learning approach instead of the weighted data [68]. It can take boolean values (true or false).

There are no dependencies/constraints between the parameters of AdaM1.

2.6 Vote [41,38]

The method for combining classifiers outputs by voting. **Parameters:**

– The number of base classifiers (nbc): It defines the number of base classifiers to be part of the voting process. This parameter will be set considering the interval: $\{nbc \in \mathbb{Z} \mid 2 \leq nbc \leq 5\}$. Therefore, Vote will have at least two (2) classifiers (because it does not make sense having just one classifier at the base level) and at most five (5) base classifiers composing the ensemble (because of the the computational cost to run the classifiers in an ensemble).
– The names of the classifiers ($B_1, ..., B_{nbc}$)[-B]: The names of the single-label base classifiers (with its respective parameters) to be used by the Vote algorithm. WEKA allows each base classifier to be constructed by any classification algorithm from Section 1. Thus, each B_i (where $1 \leq i \leq nbc$) may take a different algorithm from Section 1 with a different parametrization. This, of course, is not obligatory and depends on how the base classifiers in the ensemble are chosen.
– Combination rule (cr)[-R]: The rule for combining the outputs of the classifiers. It may take the following values:
  - AVG: Average of probabilities.
  - PROD: Product of probabilities.
  - MAJ: Majority voting.
  - MIN: Minimum probability.
  - MAX: Maximum probability.

  MAJ works by considering class predictions, whilst the other rules consider the class probabilities.

Dependencies/Constraints:

1. The parameter "The number of base classifiers" is strictly associated to the parameter "The names of the classifiers". While the former defines the quantity of classifies to compose the ensemble (nbc), the latter instantiates them ($B_1, ..., B_{nbc}$) with algorithms and configurations.

2.7 Stacking [69]

The method for combining several classifiers by using the stacking approach using an internal 10-fold cross-validation procedure. **Parameters:**

- The number of base classifiers (nbc): It defines the number of base classifiers to be part of the voting process. This parameter will be set considering the interval: $\{nbc \in \mathbb{Z} \mid 2 \leq nbc \leq 5\}$. Therefore, Vote will have at least two (2) classifiers (because it does not make sense having just one classifier at the base level) and at most five (5) base classifiers composing the ensemble (because of the the computational cost to run the classifiers in an ensemble).
- The names of the classifiers ($B_1, ..., B_{nbc}$)[-B]: The names of the single-label base classifiers (with its respective parameters) to be used by the Vote algorithm. WEKA allows each base classifier to be constructed by any classification algorithm from Section 1. Thus, each B_i (where $1 \leq i \leq nbc$) may take a different algorithm from Section 1 with a different parametrization. This, of course, is not obligatory and depends on how the base classifiers in the ensemble are chosen.

Dependencies/Constraints:

1. The parameter "The number of base classifiers" is strictly associated to the parameter "The names of the classifiers". While the former defines the quantity of classifies to compose the ensemble (nbc), the latter instantiates them ($B_1, ..., B_{nbc}$) with algorithms and configurations.

3 Search Space – Preprocessing algorithms from WEKA

In this section, the search space of (single label) preprocessing classification algorithms from WEKA [27] is studied. This is also done in order to extend and improve the search space of multi-label methods. Instead of using just a single-label classification (SLC) algorithm at the SLC base level, a wrapper containing preprocessing methods is firstly used and, just after that, SLC is performed.

3.1 Attribute Selection Classifier (ASC) [68]

The method that reduces the dimensionality of training and test data by performing attribute selection (using the training set only) before the data is set as input to a classifier. It is a wrapper approach for feature selection. **Parameters:**

- Classifier (c)[-W]: The classifier to be used. It can be any classification algorithm from Section 1.
- Search method (sm)[-S]: The search method for selecting the attribute subset to be used as input by the classifier. It may take two values:
  1. Best First: It searches the space of attribute subsets by greedy hill-climbing augmented with a backtracking facility.
  2. Greedy Stepwise : It performs a greedy forward search through the space of attribute subsets.

  Both methods use the evaluator "CfsSubsetEval", which evaluates the worth of a subset of attributes by considering the individual predictive ability of each attribute along with the degree of redundancy between them. Hence, ASC is conceptually equivalent to using the CFS (Correlation-based Feature Selection) attribute selection method followed by the use of the chosen classifier with the attributes selected by CFS.

There are no dependencies/constraints between the parameters of ASC.

4 Studing the search space of multi-label classification algorithms

We studied **30** multi-label and meta multi-label classification algorithms from the MEKA software [57], which are described in the following two sections. It is important to mention that most algorithms in (this version of) MEKA could define a **threshold** to perform the classification using the model's confidence outputs (typically, class probabilities). For the general multi-label context, it is in general better to optimize the threshold than simply using an arbitrary threshold of 0.5 [17,56]. This parameter (*pred_tshd*) [-*threshold*] could take the following values:

- Proportional cut method by instance (PCut1) [50]: It takes into account the label cardinality of the dataset, which is simply the average number of labels associated with each instance of this dataset. Thus, PCcut1 automatically calibrates the prediction confidence threshold, by minimizing the difference between the label cardinality of the training set and the label cardinality obtained with a given set of predicted labels – where the latter set is determined by the threshold value. This does not require access to the true predictions in the test set.
- Proportional cut method by label (PCutL): It is used to calibrate the prediction confidence threshold the same way as PCUT1, but for each label individually.
- The threshold could also take a unique real value between zero (0.0) and one (1.0) for all instances being classified. Formally, the threshold can also be defined by the following interval: $\{threshold \in \mathbb{R} \mid 0.0 < threshold < 1.0\}$.

5 Search Space – multi-label classification algorithms

5.1 Binary Relevance (BR) [64]

The standard Binary Relevance (BR) method. It creates a binary classification problem for each label and learns a model for each label individually. **Parameters:**

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in BR.

5.2 Binary Relevance – 'quick' version (BRq) [56]

BRq is a version of BR which is able to downsample the number of training instances across the binary models. It is intended for use in an ensemble (but it works in a standalone fashion as well).

Parameters:

- Base classifier (*bc*)[-W] : It can be any classifier from WEKA.

– Down-sample ratio (dsr)[-P]: It is a ratio used to reduce the number of instances across the binary models. Low values mean more removals and high values mean less removals, as BRq uses the following formula $(1 - dsr) * number\_of\_instances$ to calculate the number of instances to remove. This parameter is constrained by the interval:
$\{dsr \in \mathbb{R} \mid 0.2 \leq dsr \leq 0.8\}$.
There is no explanation about this parameter in the original paper and in other papers in the multi-label classification literature. The justification – about the used interval – is that we would like to have (at least) 20% of the instances from the original data to construct the model (otherwise, the method may not have sufficient instances to build the model). Additionally, we would like to have (at most) 80% of the instances from the original data in order to learn the classifier (otherwise, the method would be very similar to BR).

There are no dependencies/constraints between the parameters of BRq.

5.3 Classifier Chains (CC) [56]

The Classifier Chains (CC) method is also similar to BR, but the label outputs predicted by a classifier become new inputs for the next classifiers in the chain. It uses a single random order of labels in the chain.

Parameters:

– Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in CC.

5.4 Classifier Chains – 'quick' version (CCq) [56]

CCq is a version of CC which is able to down-sample the number of training instances across the binary models. It is also intended for use in an ensemble (but it works in a standalone fashion as well). **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Down-sample ratio (dsr)[-P]: It is a ratio used to reduce the number of instances across the binary models. Low values mean more removals and high values mean less removals, as CCq uses the following formula $(1 - dsr) * number\_of\_instances$ to calculate the number of instances to remove. This parameter is constrained by the interval:
$\{dsr \in \mathbb{R} \mid 0.2 \leq dsr \leq 0.8\}$.
There is no explanation about this parameter in the original paper and in other papers in the multi-label classification literature. The justification – about the used interval – is that we would like to have (at least) 20% of the instances from the original data to construct the model (otherwise, the method may not have sufficient instances to build the model). Additionally, we would like to have (at most) 80% of the instances from the original data in order to learn the classifier (otherwise, the method would be very similar to CC).

There are no dependencies/constraints between the parameters of CCq.

5.5 Bayesian Classifier Chains (BCC) [70]

It creates a maximum spanning tree based on marginal label dependences and then employs a classifier chain (CC). The original paper used Naïve Bayes as a base classifier, which defined the name of the method, but other types of classifiers can be used. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.
- Dependency type (dp)[-X]: The way to measure and find the dependencies. It may take ten categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf (Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. No label dependence; 10. L (The "LEAD" method for finding conditional dependence).

There are no dependencies/constraints between the parameters of BCC.

5.6 (Bayes Optimal) Probabilistic Classifier Chains (PCC) [14]

PCC acts exactly like CC at training time, but explores all possible paths as inference at test time (hence, "Bayes optimal"). **Parameters:**

- Base classifier (bc)[-W]: It can be any classifier from WEKA.

Dependencies/Constraints:

1. PCC has poor scalability, i.e., it is very slow when the number of labels is greater than a certain threshold. In the PCC's original paper [14], it is said that this threshold should be 15 labels. In the future, we might consider it to scale the proposed solution to evolve multi-label learning algorithm. For instance, we must impose a constraint in the grammar that specifies the use of PCC only if the number of labels is less than 15. We can also specify a time budget for all MLC algorithms, depending on the size of the dataset. This will consequently limit the effectiveness of the PCC algorithm in more complex types of data.

5.7 Monte-Carlo Classifier Chains (MCC and M2CC) [51,52]

These algorithms (MCC and M2CC) apply classifier chains with Monte Carlo optimization, using a maximum number of inference and chain-order trials. MCC has a tractable label prediction scheme only at the test time (MCC), whereas M2CC performs an additional search for the optimal chain sequence at the training time. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.
- Inference Iterations (ii)[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval: $\{ii \in \mathbb{Z} \mid 1 < ii \leq 100\}$.

– Chain Iterations (chi)[-Is]: The number of iterations to search the chain space at training time. This parameter is bounded by the values in the interval: $\{chi \in \mathbb{Z} \mid 1 < chi \leq 1500\}$. It can also take the value zero and the MCC algorithm is used instead of M2CC. This will happen with 50% of probability, i.e., MCC and M2CC have the same chances of being selected.
– Payoff function (pof)[-P]): It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss limited by the number of labels; 13. Log loss limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18. F_1 micro averaged; 19. F_1 macro averaged by example; 20. $F1$ macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23. Levenshtein distance.

There are no dependencies/constraints between the parameters in MCC and M2CC. Additionally, we studied the range of the parameters in the works of Read *et al.* [51,52]. However, the authors did not employ a proper parameter tuning at the single-label level, neither at the multi-label level. In the work of Read *et al.* [52], a search is performed to find the proper number of chain iterations in accordance to the payoff function. We are using part of this study to define the range of the parameters.

5.8 Population of Monte-Carlo Classifier Chains (PMCC) [51,52]

PMCC is similar to MCC and M2CC. It is considered an extension of both methods. The difference is that PMCC creates a population of M chains at training time (from Is candidate chains, using Monte Carlo sampling), and uses all of them at test time. This is not a typical majority-vote ensemble method. The simulated annealing search [37] can also be applied to the chain structures (produced by MCC or M2CC) in order to find the best one.

Parameters:

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Inference Iterations (ii)[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval: $\{ii \in \mathbb{Z} \mid 1 < ii \leq 100\}$.
– Chain Iterations (chi)[-Is]: The number of iterations to search the chain space at training time. This parameter is bounded by the values in the interval: $\{chi \in \mathbb{Z} \mid 50 < chi \leq 1500\}$.
– Beta (β)[-B]: It sets the factor with which the temperature (and thus the acceptance probability of steps in the wrong direction in the search space) is decreased in each iteration of the simulated annealing search. This parameter is bounded by the interval: $\{\beta \in \mathbb{Z} \mid 0.01 \leq \beta \leq 0.99\}$.
– Temperature switch *(ts)*[-O]: It sets the use of simulated annealing search and, when it is activated, it cools the chain down over time (from the beginning of the chain). It may take the values zero (0) or one (1). The value

zero (0) means that no temperature is used, i.e., the parameter β is ignored internally by PMCC. If using $ts = 1$, this sets the use of the β constant.
- Population size (ps)[-M]: It sets the population size. It should be always smaller than the total number of chains evaluated (Is). This parameter takes one of the values defined by the following interval:
  $\{ps \in \mathbb{Z} \mid 1 \leq ps \leq 50\}$.
- Payoff function (pof)[-P]): It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss limited by the number of labels; 13. Log loss limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18. F_1 micro averaged; 19. F_1 macro averaged by example; 20. $F1$ macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23. Levenshtein distance.

Dependencies/Constraints:

1. The parameter "population size" must be smaller than the parameter "chain iterations".

Again, we studied the range of the parameters in the works of Read *et al.* [52]. However, the authors did not employ a proper parameter tuning at the single-label level, neither at the multi-label level. During the work of Read *et al.* [52], a search is performed to find the proper number of chain iterations in accordance to the payoff function. We are using part of this study to define the range of the parameters. Nevertheless, parameters β, temperature and population size are not properly studied for the multi-label scenario.

5.9 Classifier Trellis (CT) [53]

Classifier chains in a trellis structure (rather than a cascaded chain). It is possible to set the width and type/connectivity of the trellis, and optionally to change the payoff function which guides the placement of nodes (labels) within the trellis.
Parameters:

- Base classifier (bc)[-W] : It can be any classifier from WEKA.
- Width (w)[-H]: it determines the width of the trellis (0 for chain, i.e., $w = L$; -1 for a square trellis, i.e., $w = \sqrt{L}$, always using the default floor function to convert it to an integer value). Thus, the trellis structure will always have w rows and L nodes, in total, connected using directed edges.
- Dependency type (dp)[-X]: The way to measure and find the label dependencies. It may take nine categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf (Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. No label dependence.

– Inference Iterations (ii)[-Iy]: The number of iterations to search the output space at test time. This parameter is bounded by the values in the interval: $\{ii \in \mathbb{Z} \mid 1 \leq ii \leq 100\}$.
– Chain Iterations (chi)[-Is]: The number of iterations to search the chain space at train time. This parameter is bounded by the values in the interval: $\{chi \in \mathbb{Z} \mid 1 < chi \leq 1500\}$.
– Density (d)[-L]: It determines the neighborhood density (the number of neighbors for each node in the trellis). The default value for the density parameter is one (1), and zero (0) indicates a BR classifier. Thus, this parameter is not allowed to take the value zero, being restricted by the interval: $\{d \in \mathbb{Z} \mid 1 \leq d \leq \sqrt{L} + 1\}$, where L is the total number of labels.
– Payoff function (pof)[-P]): It sets the payoff function to evaluate the chains when performing the search. It can take 23 values: 1. Accuracy; 2. Jaccard index; 3. Hamming score; 4. Exact match; 5. Jaccard distance; 6. Hamming loss; 7. Zero One loss; 8. Harmonic score; 9. One error; 10. Rank loss; 11. Average precisio; 12. Log Loss limited by the number of labels; 13. Log loss limited by the number of instances; 14. Micro Precision; 15. Micro Recall; 16. Macro Precision; 17. Macro Recall; 18. F_1 micro averaged; 19. F_1 macro averaged by example; 20. $F1$ macro averaged by label; 21. AUPRC macro averaged; 22. AUROC macro averaged; 23. Levenshtein distance.

Dependencies/Constraints:

1. If the width $w = L$ ($w = 0$), the density $d = 1$. Otherwise, if $w = \sqrt{L}$ (w = -1), the density d should be $\sqrt{L} + 1$, at most, i.e., $d \leq \sqrt{L} + 1$.

5.10 Conditional Dependency Networks (CDN) [26]

CDN builds a fully connected undirected network, where each node (label) is connected to each other node (label). Each node is a binary classifier that predicts $p(y_j|x, y_1, ..., y_{j-1}, ..., y_L)$. Then, inference is done using the Gibbs Sampling method over I iterations. Additionally, the final I_c iterations are used to collected the marginal probabilities, which become the prediction ($y[]$).

Parameters:

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Iterations (i)[-I]: The total number of iterations to perform in CDT. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 100 < i \leq 1000\}$.
– Collection iterations (ci)[-Ic] The number of collection iterations used to compute the output class probabilities in the Gibbs Sampling method. The parameter ci is restricted by the interval: $\{ci \in \mathbb{Z} \mid 1 \leq ci \leq 100\}$.

Dependencies/Constraints:

1. The collections will happen just after $(i - ci)$ iterations. So, i should be substantially greater than ci in order to make the algorithm works properly.

5.11 Conditional Dependency Trellis (CDT) [53,26]

CDT is similar to CDN. However, it constructs a trellis structure (like CT) instead of a fully connected network. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.
- Width (w)[-H]: it determines the width of the trellis (0 for chain, i.e., $w = L$; -1 for a square trellis, i.e., $w = \sqrt{L}$, always using the default floor function to convert it to an integer value). Thus, the trellis structure will always have w rows and L nodes, in total, connected using directed edges.
- Dependency type (dp)[-X]: The way to measure and find the label dependencies. It may take nine categorical values: 1. C (co-occurrence counts); 2. I (mutual information); 3. Ib (mutual information using binary approximation); 4. Ibf (Mutual information using fast binary approximation); 5. H (Conditional information); 6. Hbf (Conditional information using fast binary approximation); 7. X (Chi-squared); 8. F (Frequencies); 9. None (Using empty).
- Density (d)[-L]: It determines the neighborhood density (the number of neighbors for each node in the trellis). The default value for the density parameter is one (1), and zero (0) indicates a BR classifier. Thus, this parameter is not allowed to take the value zero, being restricted by the interval: $\{d \in \mathbb{Z} \mid 1 \leq d \leq \sqrt{L} + 1\}$, where L is the total number of labels.
- Iterations (i)[-I]: The total number of iterations to perform in CDT. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 100 < i \leq 1000\}$.
- Collection iterations (ci)[-Ic] The number of collection iterations used to compute the output class probabilities in the Gibbs Sampling method. The parameter ci is restricted by the interval: $\{ci \in \mathbb{Z} \mid 1 \leq ci \leq 100\}$.

Dependencies/Constraints:

1. If the width $w = L$ ($w = 0$), the density $d = 1$. Otherwise, if $w = \sqrt{L}$ (w = -1), the density d should be $\sqrt{L} + 1$, at most, i.e., $d \leq \sqrt{L} + 1$.
2. The collections will happen just after $(i - ci)$ iterations. So, i should be substantially greater than ci in order to make the algorithm works properly.

5.12 Four-class pairWise classification (FW) [57]

FW trains a multi-class base classifier for each pair of labels. Thus, the number of classifiers is $\frac{(L*(L-1))}{2}$ in total (where L is the number of labels), each one with four possible class values (00,01,10,11) representing the possible combinations of relevant (1)/irrelevant (0) values for each label in the label pair. It uses a voting and a threshold scheme at testing time where, e.g., 01 from pair jk gives one vote to label k and any label with a number of votes above the threshold is considered relevant. It uses the same threshold specified in the Section 4 to define the relevance of a label. **Parameters:**

- Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in FW.

5.13 The Ranking and Threshold method (RT) [50]

RT duplicates each multi-labeled example, and assigns one of the labels (only) to each copy. After that, it trains a regular multi-class base classifier. At test time, a threshold separates relevant from irrelevant labels using the posterior probability for each class value (i.e., label). It uses the same threshold specified in the Section 4 to define the relevance of a label. **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in RT.

5.14 Label Combination (LC) [64]

Label Combination (LC), also knows as Label Powerset (LP), treats each label combination as a single class in a multi-class learning scheme. The set of possible values of each class is the powerset of the set of labels.

– Base classifier (bc)[-W] : It can be any classifier from WEKA.

There are no dependencies/constraints in LC.

5.15 The Pruned Sets method (PS) [50,55]

PS was created to use the power of LC's labelset-based paradigm, without the disadvantages of such method. In order to to this, this algorithm has two important steps: a pruning step and a label-set subsampling step. The pruning step removes infrequently occurring label sets from the training data. This removes unnecessary complexity from the LC-transformed data by reducing the number of labelsets. Nevertheless, PS does not simply discard the pruned examples. Instead of doing that, PS subsamples the labelsets of these examples for label subsets which occur more frequently in the training data. It then attaches these label sets to the example, creating new examples and reintroducing them into the training. It subsamples these labelsets pv times to produce pv new examples, where pv is the pruning value (defined in the followed items).

After these steps, it trains a standard LC classifier. The idea of the method is to reduce the number of unique class values that would otherwise need to be learned by LC. PS achieves its best performance when used in an Ensemble (e.g., EnsembleML). **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Pruning value (pv)[-P]: It defines an infrequent labelset as one which occurs less than p times in the data. $p = 0$ would mean that LC classifier is learned. Thus, this parameter is bounded by the following interval: $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$.

– Subsampling value (sv)[-N]: The label set of each pruned example (in accordance to the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The PS method subsamples the label sets of pruned examples to create examples which do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval: $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$.

There are no dependencies/constraints between the parameters of PS. Additionally, there is a proper study in the work of Read [50] about the range of these two parameters.

5.16 The Pruned Sets with threshold method (PSt) [50,55,49]

Pruned Sets with a threshold, which is a modification of PS that can form new label sets at classification (i.e., test) time by using a threshold function. Given the posterior of the label classes (combinations) and the number of labels, it returns the distribution across labels. Using the threshold (defined in the Section 4) could make the method to predict labelsets not seen in the training set, differently from PS. **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Pruning value (pv)[-P]: It defines an infrequent labelset as one which occurs less than p times in the data. $p = 0$ would mean that LC classifier is learned. Thus, this parameter is bounded by the following interval: $\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$.
– Subsampling value (sv)[-N]: The label set of each pruned example (in accordance to the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The PSt method subsamples the label sets of pruned examples to create examples which do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval: $\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$.

There are no dependencies/constraints between the parameters of PSt. Additionally, there is a proper study in the work of Read [50] about the range of these two parameters (pv and sv). The parameters are the same of PS. The main thing that is changed in PSt when compared to PS occurs at the test time.

5.17 RAndom k-labEL Pruned Sets (RAkEL) [65,50]

RAkEL randomly draws M subsets of labels, each with k labels, from the set of labels, and trains PS upon each one. Finally, it combines label votes from the PS classifiers to get a label-vector prediction. **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.

– Pruning value (pv)[-P]: It prunes an infrequent labelset when it occurs less than pv times in the data. $pv = 0$ means that LC classifier is learned. Thus, this value is not allowed for RAkEL, which makes this parameter being bounded by the following interval:
$\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$.
– Subsampling value (sv)[-N]: The label set of each pruned example (in accordance to the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. This version of RAKEL in MEKA subsamples the label sets of pruned examples to create examples which do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:
$\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$.
– Number of labels for each subset (les)[-k]: It defines the number of labels in each label subset. This parameter should be bounded by the interval [45]:
$\{les \in \mathbb{Z} \mid 1 \leq les \leq \frac{L}{2}\}$, where L is the number of labels.
– Number of subsets to run in an ensemble (sre)[-M]): This parameter controls the number of models to build in a ensemble and take values in accordance to the following interval [45]:
$\{sre \in \mathbb{Z} \mid 2 \leq sre \leq min(2 \cdot L, 100)\}$, where L is the number of labels.

There are no dependencies/constraints between the parameters of RAkEL. Additionally, we followed the work of Read [50] about the range of the subsampling and pruning values. The other two parameters (number of labels in each subset and number of models to build in a ensemble) were defined in accordance to the work of Madjarov *et al.* [45].

5.18 RAndom k-labEL Disjoint Pruned Sets (RAkELd) [65,50]

RAkELd takes a random partition of labels, but unlike RAkEL the labelsets are disjoint/non-overlapping subsets. **Parameters:**

– Base classifier (bc)[-W] : It can be any classifier from WEKA.
– Pruning value (pv)[-P]: It prunes an infrequent labelset when it occurs less than p times in the data. $pv = 0$ means that LC classifier is learned. Thus, this value is not allowed for RAkEL, which makes this parameter being bounded by the following interval:
$\{pv \in \mathbb{Z} \mid 1 \leq pv \leq 5\}$.
– Subsampling value (sv)[-N]: The label set of each pruned example (in accordance to the examples pruned by the use of the previous parameter, i.e., the pruning value) becomes a candidate for label-set subsampling. The version of RAKEd in MEKA subsamples the label sets of pruned examples to create examples which do meet the pruning criterion. So, the subsample value defines the (maximum) number of frequent labelsets to subsample from the infrequent labelsets. This parameter is bounded by the following interval:
$\{sv \in \mathbb{Z} \mid 0 \leq sv \leq 5\}$.

– Number of subsets to run in an ensemble (sre)[-M]): This parameter controls the number of models to build in a ensemble and take values in accordance to the following interval [45]:
$\{sre \in \mathbb{Z} \mid 2 \leq sre \leq min(2 \cdot L, 100)\}$, where L is the number of labels.

There are no dependencies/constraints between the parameters of RAkELd. Additionally, we followed the work of Read [50] again to set the range of the subsampling and pruning values. The other two parameters (number of labels in each subset and number of models to build in a ensemble) were defined in accordance to the work of Madjarov *et al.* [45].

5.19 Multi-Label Back Propagation Neural Network (ML-BPNN) [71,54]

This is a standard Back-Propagation Neural Network [58] with multiple outputs that correspond to multiple labels. That is, each node in the output layer corresponds to a different class label. **Parameters:**

– Number of epochs (ne)[-E]: It is the number of iterations to train the neural network. It is restricted by the interval: $\{ne \in \mathbb{Z} \mid 10 \leq ne \leq 1000\}$.
– Number of hidden units (nhu)[-H]: It defines the number of hidden units in the neural network. It is import to mention that the version of ML-BPNN in MEKA is limited to one hidden layer with nhu hidden units. This parameter takes values in proportion to the number of attributes (received as input). Thus, the number of hidden units of the network can vary from 20% to 100% of the number of attributes: $\{nhu \in \mathbb{Z} \mid 0.2 \cdot number\_of\_attributes \leq nhu \leq number\_of\_attributes\}$. The proportion will always be rounded to the nearest integer.
– Learning rate (lr)[-r]: The amount by which the weights are updated during training. It is restricted by the interval:
$\{lr \in \mathbb{R} \mid 0.001 \leq lr \leq 0.1\}$.
– Momentum (m)[-m]: It is applied to the weights during updating. It is restricted by the interval: $\{m \in \mathbb{R} \mid 0.2 \leq m \leq 0.8\}$.

There are no dependencies/constraints between the parameters of ML-BPNN. Additionally, the range of values for the parameters number of epochs, momentum and learning rate were set following the work of Read and Perez-Cruz [54]. The only parameter which was defined based on a different work [71] was the number of hidden units, nhu.

5.20 Multi-Label Deep Back-Propagation Neural Network (ML-DBPNN) [31,54]

ML-DBPNN can use Restricted Boltzmann Machines (RBM) or Stacked Restricted Boltzmann Machines (DBM) to pre-train the network and, after that, it plugs in the multi-label BPNN (ML-BPNN). It is important to mention that DBM are like RBM, but with multiple layers which are trained greedily. **Parameters:**

- Number of epochs (ne)[-E]: It is the number of iterations to train the neural network. It is restricted by the interval: $\{ne \in \mathbb{Z} \mid 10 \leq ne \leq 1000\}$.
- Number of hidden units (nhu)[-H]: It defines the number of hidden units in the neural network. It is import to mention that the version of ML-DBPNN in MEKA is limited to one hidden layer with nhu hidden units. This parameter takes values in proportion to the number of attributes (received as input). Thus, the number of hidden units of the network can vary from 20% to 100% of the number of attributes: $\{nhu \in \mathbb{Z} \mid 0.2 \cdot number\_of\_attributes \leq nhu \leq number\_of\_attributes\}$. The proportion will always be rounded to the nearest integer.
- Learning rate (lr)[-r]: The amount by which the weights are updated during training. It is restricted by the interval: $\{lr \in \mathbb{R} \mid 0.001 \leq lr \leq 0.1\}$.
- Momentum (m)[-m]: It is applied to the weights during updating. It is restricted by the interval: $\{m \in \mathbb{R} \mid 0.2 \leq m \leq 0.8\}$.
- Number of layers in the DBM ($ldbm$)[-N]: It determines the number of layers of the Stacked Restricted Boltzmann Machines (DBMs) in the DBPNN. It should be a small number, such as the values in the interval: $\{ldbm \in \mathbb{Z} \mid 1 \leq ldbm \leq 5\}$. If the chosen value for $ldbm$ is equal to one (1), ML-DBPNN will use a RBM instead of a DBM.
- Multi-label Neural Network ($ml\text{-}nn$)[-W]: The Multi-Label Back Propagation Neural Network (ML-BPNN) of the Section 5.19 with its respective parameters. ML-DBPNN uses a ML-BPNN as a base classifier, using a RBM or a DBM to pre-train the network, i.e., to initialize the weight matrices. After this step, a ML-BPNN is used to train the data with the initialized weight matrices.

There are no dependencies/constraints between the parameters of ML-DBPNN. Additionally, the parameters number of epochs, momentum and learning rate were also set following the work of Read and Perez-Cruz [54]. The only parameter which was defined based on a different work [71] was the number of hidden units, nhu.

It important to mention that there is not a proper study on how to set the number of RBMs. The original paper for the application of deep learning in multi-label classification [54] just sets the use of two RBMs, i.e., the work did not considered other values. Here, we are trying to set this parameter based on a range with small integer values (from one to five).

6 Search Space – multi-label meta classification algorithms

In this section, we describe the search space of multi-label meta-algorithms in MEKA. It is important to say that some of the multi-label classifiers (presented in the last section) do not perform very well when used as the multi-label base classifier in a meta classifier. This is due to the poor scalability of such combination (meta multi-label and base multi-label). Examples of methods that would not scale up well are: MCC, PCC, PMCC, CDN and CDT (these two methods involve Gibbs sampling, which may be too expensive in an ensemble), RAkEL and RAkELd (these two methods are ensembles by themselves, and using an ensemble as base classifier would lead to a very slow ensemble of ensembles). This must be considered in the grammar or directly in the execution of the algorithm (i.e, setting a time budget for such algorithms when they are used at the multi-label base level).

6.1 Meta Binary Relevance (MBR) [25]

MBR is just the Binary Relevance (BR) method stacked with features which are label outputs, i.e., the label predictions of a BR method become the features for a new BR. **Parameters:**

- The BR method (Section 5.1) parameterized with the single-label classification algorithm, which can be an classification algorithm from Sections 1 and 2.

6.2 Subset Mapper (SM) [60]

SM maps the output of a multi-label classifier to a known label combination using the Hamming distance, i.e., it checks what label combination (label subsets) from the training set has the closest distance to the predicted label combination on the test instance using probability distribution of the label subset for this instance. In order to do that, SM transforms the probability distribution array of the label subset in a binary array. For each label subset in the training set (also represented by a binary array), it calculates the Hamming distance to the binary probability distribution array, outputting the closest label subset to the predicted distribution array. SM will map this label subset to this particular test instance. **Parameters:**

- Multi-label classifier(*mlc*)[-W]: The multi-label method that creates a model at the multi-label classification level.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5.

6.3 Random Subspace Multi-Label method (RSML)

RSML combines several multi-label classifiers in an ensemble where the attribute space and the instance space used for building each model are random subsets from the original space. In other words, RSML subsamples the attribute space and instance space randomly for each ensemble member. Basically, it is a generalized version of Random Forests. Additionally, it is computationally cheaper than EnsembleML for the same number of models in the ensemble and the same value of bag size percent.

Parameters:

- Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Bag size percent (bsp)[-P]: The size of the bag in percentage of the training set size (number of training instances), and it is defined by the interval: $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$.
- Number of iterations (i)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$.
- Attribute percent (ap)[-A]: The size of the attribute space, as a percentage of total attribute space size (number of attributes). This parameter is bounded by the following interval: $\{ap \in \mathbb{Z} \mid 10 \leq ap \leq 100\}$.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5.

The range of the number of iterations for this ensemble method was defined based on Read's thesis [56]. The range of values for this parameter also considers scalability issues as we need to run a multi-label algorithm many times in an ensemble. The parameter "bag size percent" is defined in the MEKA documentation. The attribute percentage was set in accordance to the single-label version for the same method. This was done because there is not any work that studies this algorithm for multi-label classification.

6.4 Multi-Label Classification using Boolean Matrix Decomposition (MLC-BMaD) [67]

MLC-BMaD transforms the labels using a Boolean matrix decomposition. The first resulting matrix is used as latent labels and a classifier is trained to predict them. The second matrix is used in a multiplication to decompress the predicted latent labels. **Parameters:**

- Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Size (s)[-size]: It determines the size of the compressed matrix. It can take a value in the following interval: $\{s \in \mathbb{Z} \mid 1 \leq s \leq L\}$, where L is the number of labels.

– Threshold ($tshd$)[-threshold]: It defines the threshold for the matrix decomposition. The threshold sets the minimum frequency of a label pair to be considered a frequent co-occurrence. More precisely, this parameter defines the association between two labels. So, if the calculated degree of association (confidence) between two labels is smaller than the threshold, there is no association between them. Otherwise, there is one. This parameter varies between 0.0 (all are frequent) and 1.0 (must be in all rows to be frequent). More formally, it is restricted by the interval: $\{tshd \in \mathbb{R} \mid 0.0 \leq tshd \leq 1.0\}$.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5.

6.5 Bagging of Multi-Label methods (BaggingML)

BaggingML combines several multi-label classifiers using Bootstrap AGGregatING (Bagging) [8]. It randomly sets weights higher than zero to certain instances, on only those instances are chosen for the bag. The parameter "bag percent size" is then not used as the number of instances in the bag is just based on the weight values. Thus, the members of the ensemble could have 100% of the instances if all of them have a weight assigned. **Parameters:**

– Multi-label classifier(mlc)[-W]: The multi-label method that creates a model at the base multi-label classification level.
– Number of iterations (i)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$. The range of the number of iterations for this ensemble method was defined by Read's thesis [56].

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5, except for BCC, which is not suitable for this meta-learner.

6.6 Bagging of Multi-Label methods with Duplicates (BaggingMLDup)

BaggingMLDup also combines several multi-label classifiers using Bootstrap AGGregatING. However, it uses the parameter "bag size percent" to define a specific number of instances for each member (classifier) of the ensemble. After that, it randomly samples instances, being able to sample the same instance (*duplicates*) for the bag. This method does not use any weight to select the instances for the members of the ensemble. **Parameters:**

– Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the multi-label classification level.

– Bag size percent (bsp)[-P]: The size of the bag in percentage of the training set size (number of training instances) and it is defined by the interval: $\{bsp \in \mathbb{Z} \mid 10 \leq bsp \leq 100\}$.
– Number of iterations (i)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5, except for BCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined by Read's thesis [56]. The parameter "bag size percent" is defined in the MEKA documentation.

6.7 Ensemble of Multi-Label methods (EnsembleML)

EnsembleML combines several multi-label classifiers in a simple-subset ensemble. This method is very similar to BaggingMLDup. The only difference is that BaggingMLDup allows sampling with replacement for each model, whereas EnsembleML uses sampling without replacement.

Parameters:

– Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the base multi-label classification level.
– Bag size percent (bsp)[-P]: The size of the bag in percentage of the training size (number of training instances) and it is defined by the interval: $\{bsp \in \mathbb{Z} \mid 52 \leq bsp \leq 72\}$.
– Number of iterations (i)[-I]: The number of iterations to perform, i.e., the number of members in the ensemble. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5, except for BCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined by Read's thesis [50]. Additionally, in his thesis, the author mentioned that they found that values around 62% are the best ones for the parameter "bag size percent" in a ensemble without replacement, which is the case. Thus, we are trying to set the range for this parameter introducing lower and upper bounds close to this value (10% smaller and 10% greater).

6.8 Expectation Maximization (EM) [15]

In EM, a specified multi-label classifier is built on the training data. This model is then used to classify the training data. The confidence with which instances are classified is used to reweight them. This data is then used to retrain the classifier. This cycle continues ('EM'-style) for I iterations. The final model is used to classify the test data. Because of the weighting, it is advised to use a classifier which gives good confidence (probabilistic) outputs. **Parameters:**

- Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the multi-label classification level.
- Number of iterations (i)[-I]: The number of iterations to perform. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$.

Dependencies/Constraints:

1. The classifier at the base multi-label classification level should be capable to produce probabilistic predictions. However, in our preliminary tests, most multi-label classification methods described in Section 5 were suitable for this meta-learner, except for PMCC. Thus, we will use all the suitable methods in accordance to these experiments at the base multi-label classification level.

The range of the number of iterations for this ensemble method was defined based on the Read's thesis [56]. The range of values for this parameter also considers scalability issues as we need to run a multi-label algorithm many times in an ensemble. This was done because there is not an appropriate work that studies this algorithm for multi-label classification.

6.9 Classification Maximization (CM)

CM trains a classifier with labeled and unlabeled data (semi-supervised) learning using the Classification Expectation algorithm, which is a hard version of EM algorithm, as it does not update the instance weights using (a product factor of) the probability distribution produced by the classifier. Instead, it sets to zero (0.0) or one (1.0) the weight of any instance in the dataset. Unlike EM, it can use any classifier, not necessarily one that gives good probabilistic outputs. **Parameters:**

- Multi-label classifier (mlc)[-W]: The multi-label method that creates a model at the base multi-label classification level.
- Number of iterations (i)[-I]: The number of iterations to perform. This parameter is restricted by the interval: $\{i \in \mathbb{Z} \mid 10 \leq i \leq 50\}$.

Dependencies/Constraints:

1. The multi-label classification method can be any one described in the Section 5, except for PMCC, which is not suitable for this meta-learner.

The range of the number of iterations for this ensemble method was defined based on Read's thesis [56]. The range of values for this parameter also considers scalability issues as we need to run a multi-label algorithm many times in an ensemble. This was done because there is not any work that studies this algorithm for multi-label classification.

6.10 Hierarchy Of Multi-label classifiERs (HOMER) [63]

The HOMER algorithm constructs a hierarchy of multi-label classifiers, each one dealing with a much smaller set of labels when compared to the total number of labels, L, and also with a more balanced example distribution. **Parameters:**

- Multi-label classifier (mlc): The multi-label method that creates a model at the base multi-label classification level.
- Type (t): It can take the values Balanced Clustering, Clustering or Random.
- Number of clusters (k): It represents the number of clusters, and it is defined by the following interval: $\{s \in \mathbb{Z} \mid 2 \leq l < L\}$, where L is the number of labels.

Dependencies/Constraints:

1. In the version of HOMER for MEKA (a wrapper from Mulan), the multi-label classification method may take only the following three algorithms: Binary Relevance(BR), Label Powerset (LC) or Classifier Chain (CC), which can be augmented with any single-label classification algorithm from WEKA.

7 Grammar

Figures 1-7 present the produced grammar that encompasses the knowledge about multi-label classification in MEKA. This grammar uses the Backus Naur Form (BNF), where each production rules has, for instance, the form <*Start*>::= <*Meta-algorithm*> <*Algorithm*>. Symbols wrapped in "< >" represent non-terminals, and the special symbols "|", "[]" and "()" represent respectively a choice, an optional element and a set of grouped elements that should be used together. Additionally, the symbol "#" represents a comment in the grammar, i.e., it is ignored by the grammar's parser. The choice of one among all elements connected by "|" is made using a uniform probability distribution (i.e., all elements are equally likely), unless mentioned otherwise.

In the first production rule (<Start>) used to describe the multi-label classification (MLC) search space, MLC-PT denotes problem transformation, MLC-AA denotes algorithm adaptation, and META-MLC-LEVEL denotes the multi-label meta-algorithms. These three types of MLC algorithms have the same probability of being chosen (33%) and must use a prediction threshold (< pred_tshd >), which defines the threshold to perform the classification using the model's confidence outputs. In addition, Majority Labelset Classifier (MALC) can be selected with a different and small probability (1%), because this is a very naïve method and usually does not present good classification performances.

The grammar rule defining the problem transformation methods, i.e., <MLC-PT>, has two components in the right-hand side, namely the actual problem transformation algorithm <ALGS-PT> (defined in Figure 5) and the single-label classification algorithm (SLC, which is represented by the rule <ALG-SLC> in the grammar) to perform the single-label classification task(s). This happens because the problem transformation method transforms the multi-label task into one or more single-label tasks. We start discussing the rule defining <ALG-SLC>.

We divided the SLC algorithms in six (6) types for the grammar following the WEKA software: Trees, Rules, Lazy, Functions, Bayes and Exceptions. The last type was created just to simplify the grammar. Figure 1 shows the grammar rules for Tree algorithms. Figure 2 shows the grammar rules for Rules and Lazy algorithms. Figure 3 shows the grammar rules for the other three types of SLC algorithms. Figure 1 also defines the Attribute Selection Classifier (ASC), a wrapper which can be used together with the SCL algorithms. In this case, a preprocessing method is used before the classification step is performed.

It is also important to mention that some methods at the single-label level, such as Decision Stump and ZeroR, do not have user-defined parameters. Others, such as the Bayesian Network Classification algorithms, do not have user-defined parameters in the Auto-WEKA software, even though they have user-defined parameters in WEKA. That is, the developers of Auto-WEKA have chosen to use a fixed predefined number of parameter settings for some algorithms. As we are following Auto-WEKA to define the parameters at this level, the absence of user-defined parameters in some methods was maintained.

At the single-label level, we also have meta-algorithms, divided in four (4) types: META1, META2, META3 and META4. These four categories of meta-

algorithms are firstly called in Figure 1. As shown in Figure 4, META1 may take the two (2) meta-algorithms AdaM1 and Locally Weighted Learning, that need a base classifier at the SLC level that handles weighted instances. This is the reason the rule <ALG-WEIGHTED-TYPE> is defined. On the other hand, META2 may take just one algorithm, i.e., Random Committee. The reason for that is because this SLC meta-algorithm can only be used with randomizable base classifiers. The rule <ALG-RANDOM-TYPE> expresses these randomizable classifiers. The least restricted meta-algorithms are Random Subspace and Bagging, which are specified by META3, being able to use any SLC base classifier (from <ALG-TYPE>). Contrarily, meta-algorithms from META4 may take various base algorithms (at least two and at most five) at the SLC level (see the third rule in Figure 1). As shown in the two last rules of Figure 4, we have two methods for combining the predictions of members of the ensemble in this second type of meta-algorithm, namely Stacking and Vote.

The second component of problem transformation methods is the actual problem transformation algorithm to deal with the single-label classification. In other words, this step defines the choice of the MLC algorithm to handle the results created by the single-label classification models. For this component, we divided its respective algorithms into four (4) categories, i.e., four production rules in the grammar (see Figure 5): <ALGS-PT1>, <ALGS-PT2>, <ALGS-PT3> and <ALGS-PT4>. The main reason for the creation of these (sub-)categories is related to the constraints of the multi-label meta-algorithms in the MEKA software. Although all MLC algorithms can be used in a standalone fashion, they can also be combined with multi-label meta-algorithms. In MEKA, some MLC algorithms work very well at the multi-label base level of meta-algorithms, whereas others do not. Thus, we had to create rules in the grammar to overcome the limitations in the used software. The next paragraphs will refer to the Figures 5 and 7 to explain these links and constraints between problem transformation algorithms and multi-label meta-algorithms.

The production rule <ALGS-PT1> in Figure 5 was created to contain the algorithms Binary Relevance (BR), Classifier Chain (CC) and Label Powerset (LP). These three algorithms are the only ones which can be combined with HOMER meta-algorithm, that is defined by the production rule <META-MLC4> in Figure 7. The production rule <ALGS-PT1> is also found in the following rules in Figure 7: <META-MLC1> (via <ALGS-PT>), <META-MLC2> and <META-MLC3>. Nevertheless, the rule <META-MLC4> in Figure 7 uses it exclusively because of a limitation of the MEKA software. This is why such production rule was created in the grammar.

We referred to the second production rule to define problem transformation methods as <ALGS-PT2> in Figure 5. This rule encompasses the quick versions of BR and CC (i.e, BRq and CCq), all the complex classifier chains and trellis algorithms (which are defined by the rule <ComplexCC_Trellis>), Four-class pairWise (FW), Ranking and Threshold (RT), and all the label powerset based algorithms (which are defined by the rule <LP_based>). The production rule <ALGS-PT2> is present in the following rules in Figure 7: <META-MLC1> (via <ALGS-PT>), <META-MLC2> and <META-MLC3>. This means that

this category of PT methods describes the majority of the MLC algorithms in MEKA (57.14% of the cases, i.e., 12 of the 21 MLC algorithms) and, in addition, all these algorithms can be combined with most meta-algorithms in the MEKA software (81.82% of the cases, i.e., 9 of the 11 MLC meta-algorithms), except for HOMER (as explained previously), and Meta Binary Relevance (MBR). Thus, <ALGS-PT2> can be considered the least restrictive of the PT method rules in the grammar.

<ALGS-PT3>, in Figure 5, is the production rule to describe solely the Bayesian Classifier Chain (BCC) algorithm, one of the most constrained algorithms in the MEKA software. The BCC algorithm can only be executed in a standalone fashion or combined with the algorithms described by the production rules <META-MLC1> (via <ALGS-PT>) and <META-MLC3>. This means that BCC can be used with five (5) of the 11 meta-algorithms (in Figure 7): Subset Mapper (SM), Random Subspace Multi-Label (RSML), MLC using Boolean Matrix Decomposition (MLC-BMaD), Expectation Maximization (EM) and Classification Maximization (CM). In other cases of trying to use BCC, this will result in errors in MEKA's output and, therefore, this was not allowed in the grammar.

Similarly to <ALGS-PT3>, we have <ALGS-PT4>, a problem transformation rule that represents the Population of Monte-Carlo Classifier Chains (PMCC) algorithm. This algorithm can only be used by itself and at the multi-label base level of six (6) of the 12 meta-algorithms: Subset Mapper (SM), Random Subspace Multi-Label (RSML), MLC using Boolean Matrix Decomposition (MLC-BMaD), Bagging of Multi-Label methods (BaggingML), Bagging of Multi-Label methods with Duplicates (BaggingMLDup) and Ensemble of Multi-Label methods (EnsembleML). These six multi-label meta-algorithms are defined by the production rules <META-MLC1> and
<META-MLC2>. Therefore, the creation of <ALGS-PT3> is justified by the fact that PMCC algorithm can only be combined with these meta-algorithms, i.e., a constraint that did not appear in the other rules of the grammar.

The previous paragraphs explained in general lines the problem transformation methods in the grammar and the relationships of such methods with the meta-algorithms. Briefly, we determined the relationships between four categories of problem transformation methods (ALGS-PT$1-4$) and four categories of multi-label meta-algorithms (META-MLC$1-4$).

Besides the problem transformation methods, we also have a multi-label version of the back propagation algorithm for training neural networks, called ML-BPNN, and its deep version, called ML-DBPNN. These algorithms can be seen in the Figure 6 and are the only ones (for now) representing the algorithm adaptation (AA) methods, defined by the production rule <MLC-AA>. ML-BPNN can also be associated to meta-algorithms. As we can see in Figure 7, this MLC algorithm can be linked to the meta-algorithms defined by the production rules <META-MLC1>, <META-MLC2> and <META-MLC3>. It is not possible to combine ML-BPNN and HOMER (which is described by <META-MLC4>) because of a constraint of the MEKA software, which allows just BR, CC and LP as the algorithms at the multi-label base level of HOMER. We also do not allow

the ML-DBPNN at the multi-label base level of a meta-algorithm because of the high computational cost of such combination. This is indicated in the production rules <META-MLC1>, <META-MLC2> and <META-MLC3> in Figure 7 by using only ML-BPNN as possible combination for MLC meta-algorithms of these rules..

Finally, Figure 7 covers all the multi-label meta-algorithms, which are defined by the production rule <META-MLC-LEVEL>. As we explained previously, we created the production rules <META-MLC1>, <META-MLC2>, <META-MLC3> and <META-MLC4> in order to expand these four rules into <META-MLC-LEVEL> to control the limitations, constraints and dependencies of the MEKA software between meta-algorithms and multi-label algorithms (problem transformation and algorithm adaptation methods). These four rules formalize the majority of the meta-algorithms in MEKA – 90.91% of the meta-algorithms, which represents 10 of the 11 cases in Section 6. The exception is the Meta Binary Relevance (MBR), which is a meta-algorithm we have to distinguish from these four general production rules. This happened because MBR can only be used with Binary Relevance at the multi-label base level. This constraint is indicated in the first rule of Figure 7

```
<Start> ::= (<MLC-PT> | <MLC-AA> | <META-MLC-LEVEL> ) <pred_tshd>

<MLC-PT> ::= <ALGS-PT> <ALGS-SLC>

<ALGS-SLC> ::= <ALG-TYPE> | <META1> <ALG-WEIGHTED-TYPE> | <META2> <ALG-RANDOM-TYPE> |
               (<META3> | <META4> [<ALG-TYPE>] [<ALG-TYPE>] [ALG-TYPE>] <ALG-TYPE>) <ALG-TYPE>

<ALG-TYPE> ::= [ASC <sm>] (<TREES> | <RULES> | <LAZY> | <FUNCTIONS> | <BAYES> | <EXCEPTIONS>)
                                                    #ASC='Attribute Selection Classifier'

<sm> ::= GreedyStepwise | BestFirst                 #sm='search method'


<TREES> ::= <J48> | DecisionStump | ( ( (RandomForest <nt> | <RandomTree>) <nf> ) | <REPTree> ) <md>

<J48> ::= <J48-Basics> ( (<cf> [sr]) | u )          #sr='subtree raising', u='unpruned'
<J48-Basics> ::= <mno> [ct] [bs] [umc] [ul]         #ct='collapse tree', bs='binary splits'
                                                    #umc='use MDL correction', ul='use Laplace'
<cf> ::= RANDOM-REAL(0.0, 1.0)                      #cf='confidence factor'
<mno> ::= RANDOM-INT(1, 64)                         #mmo='minimum number of objects'

<nt> ::= RANDOM-INT(2, 256)                         #nt='number of trees'
<nf> ::= RANDOM-INT(2, 32) | 0                      #nf='number of features'
<md> ::= RANDOM-INT(2, 20) | 0                      #md='maximum depth'

<RandomTree> ::= <mw> <nfbgt>
<mw> ::= RANDOM-INT(1,64)                           #mw='minimum weight for instances in a leaf'
<nfbgt> ::= 0 | <growing_the_tree_and_backfitting>  #nfbgt='number of folds for back-fitting
                                                    #and for growing the tree'
<growing_the_tree_and_backfitting> ::= 2 | 3 | 4 | 5

<REPTree> ::= <mw> [up]                             #mw='minimum weight for instances in a leaf'
                                                    #up='use pruning'
                                                    #mw is not included in the same rule for Random Tree
                                                    #and for REPTree because of the grammar's constraints
```

Fig. 1. Defined Grammar – Part 1: General and SLC Trees Algorithms.

```
<RULES> ::= <DT> | <JRip> | OneR <mbs> | <PART> | ZeroR

<DT> ::= <em> [uibk] <sm> <crv>                          #uibk='use IBk'
                                                         #sm='search method -- defined earlier'
<em> ::= acc | rmse | mae | auc                          #em='evaluation measure'
<crv> ::= 1 | 2 | 3 | 4                                  #crv='number of folds for cross-validation'

<JRip> ::= <mtw> [cer] [up] <o>                          #cer='check error rate', up='use pruning'
<mtw> ::= RANDOM-REAL(1.0, 5.0)                          #mtw='minimum total weight for instances
                                                         #covered by a rule'
<o> ::= RANDOM-INT(1,5)                                  #o='number of optimization runs'

<mbs> ::= RANDOM-INT(1,32)                               #mbs='minimum bucket size'

<PART> ::= <PART-BASICS> (rep <nr> | ebp)                #rep='use reduced-error pruning'
                                                         #nr='number of folds for reduced-error pruning'
                                                         #ebp='use error-based pruning'
<PART-BASICS> ::= <mno> [bs]                             #mno='minimum number of objects'
<nr> ::= RANDOM-INT(2,5)

<LAZY> ::= <KNN> | <K*>


<KNN> ::= <k_nn> [loo] [<dw>]                            #loo='leave-one-out to set the k value given the range'
<k_nn> ::= RANDOM-INT(1,64)                              #k_nn='number of neighbors'
<dw> ::= F | I                                           #dw='distance weighting'

<K*> ::= <gb> [eab] <mm>                                 #eab='entropic auto-blending'
<gb> ::= RANDOM-INT(1,100)                               #gb='global blending'
<mm> ::= a | d | m | n                                   #mm='missing mode to deal with missing values'
```

Fig. 2. Defined Grammar – Part 2: SLC Rules and Lazy Algorithms .

```
<FUNCTIONS> ::= <VotedPerceptron> | <MultiLayerPerc> |
                (<StocGradDescent> | LogisticRegression) <r> | <SeqMinOptimization>

<VotedPerceptron> ::= <i> <mk> <e>
<i> ::= RANDOM-INT(1,10)                                 #i='number of iterations'
<mk> ::= RANDOM-INT(5000, 50000)                         #mk='maximum number of alterations to the perceptrons'
<e> ::= RANDOM-REAL(0.2, 5.0)                            #e='The exponent for the polynomial kernel'


<MultiLayerPerc> ::= <lr> <m> <nhn> [n2b] [r] [d]        #n2b='nominal to binary filter',
                                                         #r='use reset approach',
                                                         #d='decay in the learning rate'
<lr> ::= RANDOM-REAL(0.1, 1.0)                           #lr='learning rate'
<m> ::= RANDOM-REAL(0.0, 1.0)                            #m='momentum'
<nhn> ::= a | i | o | t                                  #nhl='rules to define the number of hidden nodes'

<StocGradDescent> ::= <lf> <lr_sgd> [nn] [nrmv]          #nn='do not normalize',
                                                         #nrmv='do not replace missing values'
<lf> ::= 0 | 1 | 2                                       #lf='loss function'
<lr_sgd> ::= RANDOM-REAL(0.00001, 1.0)                   #lr_sgd='learning rate for SGD'
<r> ::= RANDOM-REAL(0.000000000001,10.0)                 #r='ridge value in the log-likelihood'


<SeqMinOptimization> ::= <c> <ft> [bcm] <kernel>         #bcm='build calibration models'
<c> ::= RANDOM-REAL(0.5,1.5)                             #c='the cost, i.e.,complexity parameter'
<ft> ::= 0 | 1 | 2                                       #ft='filter type'
<kernel> ::= ( NormPolyKernel |
               PolyKernel
             ) <exp> [ulo] |                             #ulo='use lower order'
              Puk <om> <sig> | RBF <g>
<exp> ::= RANDOM-REAL(0.2, 5.0)                          #exp='the exponent'
<om> ::= RANDOM-REAL(0.1, 1.0)                           #om='the omega value'
<sig> ::= RANDOM-REAL(0.1, 10.0)                           #sig='the sigma value'
<g> ::= RANDOM-REAL(0.001, 1.0)                          #g='the gamma value'

<BAYES> ::= NaiveBayes [<NB-Parameters>] | <BayesianNetworkClassifiers> | NaiveBayesMultinomial
<NB-Parameters> ::= uke | usd                            #uke='use kernerl estimator'
                                                         #usd='use supervised distribution'

<BayesianNetworkClassifiers> ::= TAN | K2 | HillClimber | LAGDHillClimber | SimulatedAnnealing | TabuSearch

<EXCEPTIONS> ::= (SimpleLogistic [ucv] |                 #ucv='use cross-validation'
                  <LogisticModelTrees>
                 ) [uaic] [<wtb>]                        #uaic='use AIC measure as stopping criteria'
<LogisticModelTrees> ::= [cn] [sor] [fr] [eop]           #cn='convert nominal to binary'
                                                         #sor='split on residuals'
                                                         #fr='fast regression', eop='error on probabilities'
<wtb> ::= RANDOM-REAL(0.0, 1.0)                          #wtb='weight trim beta'
```

Fig. 3. Defined Grammar – Part 3: SLC Functions, Bayes and Exceptions Algorithms.

```
<META1> ::= <LWL> | <AdaM1>

<LWL> ::= [<k_lwl>] [<wk>]                              #LWL='Locally Weighted Learning'
<k_lwl> ::= -1 | 10  | 30 | 60 | 90 | 120               #k_lwl='number of neighbors in LWL'
<wk> ::= 0 | 1 | 2 | 3 | 4                              #wk='weighting kernel'

<AdaM1> ::= <wt> [ur] <ni_ada_and_bagging>              #ur='use resampling'
<wt> ::= RANDOM-INT(50, 100) | 100                      #wt='weight threshold'
<ni_ada_and_bagging> ::= RANDOM-INT(2, 128)             #ni_ada_and_bagging='number of iterations for
                                                        #AdaM1 and Bagging'


<ALG-WEIGHTED-TYPE> ::= <TREES> | <RULES-PARTIAL> | <KNN> | <BAYES> | <FUNCTIONS-PARTIAL>
<RULES-PARTIAL> ::= <DT> | <JRip> | <PART> | ZeroR
<FUNCTIONS-PARTIAL> ::= <MultiLayerPerc> |  <SeqMinOptimization> | <SimpleLogistic> <uaic> <wtb_activate>



<META2> ::= RandomCommittee <ni_random_methods>
<ni_random_methods> ::= RANDOM-INT(2, 64)               #ni_random_methods='number of iterations for
                                                        #random methods'
<ALG-RANDOM-TYPE> ::=  ( ( (RandomForest <nt> | <RandomTree>) <nf> ) | <REPTree> ) <md>  |
                       <StocGradDescent> <r> | <MultiLayerPerc>


<META3> ::= <Bagging> | <RandomSubspace>

<Bagging> ::= <bsp> | 100 coob  <ni_ada_and_bagging>   #coob='calculate out-of-bag'
                                                        #when coob is true, bag percent size must be 100
<bsp> ::= RANDOM-INT(10, 100)                           #bsp='bag size percent'
<RandomSubspace> ::= <sss> <ni_random_methods>
<sss> ::= RANDOM-REAL(0.1, 1.0)                         #sss='subspace size'


<META4> ::= Stacking | Vote <cr>                        #the base algorithms are defined in the third
                                                        #rule of the grammar
<cr> ::= AVG | PROD | MAJ | MIN | MAX                   #cr='combination rule'
```

Fig. 4. Defined Grammar – Part 4: SLC Meta-Algorithms.

```
<ALGS-PT> ::= <ALGS-PT1> | <ALGS-PT2> | <ALGS-PT3> | <ALGS-PT4>
<ALGS-PT1> ::= BR | CC | LP                                   #BR='Binary Relevance', CC='Classifier Chain'
                                                              #LC='Label Powerset'
<ALGS-PT2> ::= (BRq | CCq) <dsr> | <ComplexCC_Trellis> |      #BRq and CCq = 'quick versions for BR and CC'
               FW | RT | <LP_based>                           #FW='Four-class pairWise', RT='Ranking-Threshold'
<ALGS-PT3> ::= BCC <dp_complete>                              #BCC='Bayesian Classifier Chain'
<ALGS-PT4> ::= PMCC <B> <ts> <ii> <chi_PMCC> <ps> <pof>       #PMCC='Population of Monte-Carlo Classifier Chains'

<dsr> ::= RANDOM-REAL(0.2, 0.8)                               #dsr='down-sample ratio'

<ComplexCC_Trellis> ::= PCC | (MCC <chi_MCC> | <CT>) <ii> <pof> |
                        (CDN | <CDT>) <i_cdn_cdt> <ci>        #PCC='Probabilistic Classifier Chains'
                                                              #MCC='Monte-Carlo Classifier Chains'
                                                              #CT='Classication Trellis'
                                                              #CDN='Conditional Dependency Networks'
                                                              #CDT='Conditional Dependency Trellis'

<chi_MCC>::= <chi_CT> | 0                                     #chi_MCC='nmber of chain iterations for MCC'
<ii> ::= RANDOM-INT(2, 100)                                   #ii='number of inference interations'
<pof> ::= Accuracy | Jaccard index | Hamming score | Exact match | Jaccard distance | Rank loss |
          Hamming loss | Zero One loss | Harmonic score | Log Loss lim:L | Micro Recall | One error |
          Log Loss lim:D | Micro Precision | Macro Precision | Macro Recall | F1 micro averaged |
          Avg precision | F1 macro averaged by example | F1 macro averaged by label | AUPRC macro averaged |
          AUROC macro averaged | Levenshtein distance
                                                              #pof='Payoff function'

<CT> ::= <chi_CT> <w> <dp>
<dp> ::= C | I | Ib | Ibf | H | Hbf | X | F | None           #dp='dependency type'
<chi_CT> ::= RANDOM-INT(2, 1500)                              #chi_CT='number of chain iterations for CT'
<w> ::= 0 1 | -1 <d>                                          #w='width of the trellis'
<d> ::= RANDOM-INT(1, SQRT(L) +1)                             #d='neighborhood density'
                                                              #Where L is the number of labels
<CDT> ::= <w> <dp>                                            #parameters defined earlier

<i_cdn_cdt> ::= RANDOM-INT(101, 1000)                         #i_cdn_cdt='total number of iterations'
<ci> ::= RANDOM-INT(1, 100)                                   #ci='collection iterations'

<LP_based> ::= (PS | PSt | <RAkEL-based> ) <sv> <pv>          #PS='Pruned Sets'
                                                              #PSt='Pruned Sets with Threshold'
<sv> ::= RANDOM-INT(0, 5)                                     #sv='subsampling value'
<pv> ::= RANDOM-INT(1, 5)                                     #pv='pruning value'

<RAkEL-based> ::= (RAkEL <sre> | RAkELd)  <les>               #RAkEL='RAndom k-labEL Pruned Sets'
                                                              #RAkELd='RAndom k-labEL Disjoint Pruned Sets'
<sre> ::= RANDOM-INT(2, min(2L, 100) )                        #sre='number of subsets to run in an ensemble'
<les> ::= RANDOM-INT(1, L/2)                                  #les='number of labels in each label subset'
                                                              #Where L is the number of labels
<dp_complete> ::= <dp> | LEAD                                 #dp='complete dependency type for BCC'

<B> ::= RANDOM-REAL(0.01, 0.99)                               #B='Beta factor for deacreasing the temperature'
<ts> ::= 0 | 1                                                #ts='Temperature switch'
<ps> ::=  RANDOM-INT(1, 50)                                   #ps='population size'
<chi_PMCC> ::= RANDOM-INT(51, 1500)                               #chi_PMCC='number of chain iterations for PMCC'
```

Fig. 5. Defined Grammar – Part 5: MLC Problem Transformation Methods.

```
<MLC-AA> ::= <ML-DBPNN> <ML-BPNN> | <ML-BPNN>

<ML-BPNN> ::= <ne> <nhu_bpnn> <lr_bpnn> <m_bpnn>                  #ML-BPNN='Multi-Label Back Propagation
                                                                  #         Neural Network'
<ne> ::= RANDOM-INT(10, 1000)                                     #ne='number of epochs'
<nhu_bpnn> ::= RANDOM-REAL(0.2, 1.0) * n_attributes               #nhu_bpnn='number of hidden units, that
                                                                  #is a parameter that dependes on the
                                                                  #number of attributes of the dataset'

<lr_bpnn> ::= RANDOM-REAL(0.001, 0.1)                             #lr_bpnn='learning rate for BPNN/DBPNN'
<m_bpnn> ::=  RANDOM-REAL(0.2, 0.8)                               #m_bpnn='momentum for BPNN and DBPNN'



<ML-DBPNN> ::= <ne> <nhu_bpnn> <lr_bpnn> <m_bpnn> <rbm>           #ML-DBPNN='Deep ML-BPNN'
                                                                  #MLC-AA='ML-BPNN'
<rbm> ::= RANDOM-INT(1, 5)                                        #rbm='number of layers of (Stacked)
                                                                  #     Restricted Boltzmann Machines'
```

Fig. 6. Defined Grammar – Part 6: MLC Algorithm Adaptation Methods..

```
<META-MLC-LEVEL> ::= <META-MLC1> | <META-MLC2> | <META-MLC3> | <META-MLC4> | MBR BR <ALGS-SLC>
                                                          #MBR='BR method stacked with feature outputs'
                                                          #META-MLC 1-4='meta MLC algorithms
                                                          # with different constraints'

<META-MLC1> ::= (SM | <RSML> | <MLC-BMaD>) (<ALGS-PT> <ALGS-SLC> | <ML-BPNN>)
                                                          #SM='Subset Mapper -- MLC method as parameter'

<RSML> ::= <bsp> <i_metamlc> <ap>                         #RSML='Random Subspace Multi-Label'
<bsp> ::= RANDOM-INT(10, 100)                             #bsp='bag size percent'
<i_metamlc> ::= RANDOM-INT(10, 50)                        #i_metamlc='number of iterations for
                                                          #meta MLC methods'
<ap> ::= RANDOM-INT(10, 100)                              #ap='attribute percent'

<MLC-BMaD> ::= <s> <tshd>                                 #MLC-BMaD='MLC using Boolean Matrix Decomposition'
<s> ::= RANDOM-INT(1, L)                                  #s='size of the compressed matrix'
<tshd> ::= RANDOM-REAL(0.0, 1.0)                          #tshd='threshold for the matrix decomposition'

<META-MLC2> ::=  <alg-meta-mlc2> ((<ALGS-PT1> | <ALGS-PT2> | <ALGS-PT4>)  <ALGS-SLC> | <ML-BPNN>)

<alg-meta-mlc2> ::= ((BaggingML | BaggingMLDup <bsp> ) | EnsembleML <bsp_ensembleML>) <i_metamlc>
                                                             #BaggingML='Bagging of Multi-Label methods'
                                                             #BaggingMLDup='BaggingML with duplicates'
                                                             #EnsembleML='Ensemble of Multi-Label methods'
                                                             #bsp='bag size percent -- defined earlier'

<bsp_ensembleML> ::= RANDOM-INT(52, 72)                      #bsp_ensembleML='specific bsp for EnsembleML'

<META-MLC3> ::= ( (EM | CM ) <i_metamlc> ) ((<ALGS-PT1> | <ALGS-PT2> | <ALGS-PT3>) <ALGS-SLC> | <ML-BPNN>)
                                                                        #EM='Expectation Maximization'
                                                                        #CM='Classification Maximization'



<META-MLC4> ::= <HOMER> <ALGS-PT1> <ALGS-SLC>
<HOMER> ::= <t> <k_homer>                                 #HOMER='Hierarchy Of Multi-label classifiERs'
<t> ::= BalancedClustering | Clustering | Random          #t='the type of clustering'
<k_homer> ::= RANDOM-INT(2, L-1)                            #k_hommer='number of clusters to be created'


<pred_tshd> ::= PCut1 | PCutL | RANDOM-REAL(0.001, 0.999)   #pred_tshd='prediction threshold'
                                                          #PCut1='P-Cut method',PCutL='P-Cut method by Label'
```

Fig. 7. Defined grammar – Part 7: MLC Meta-Algorithms.

References

1. M. Abramovici, M. Neubach, M. Fathi, and A. Holland. Competing fusion for Bayesian applications. In *Proc. of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 378–385, 2008.
2. C. C. Aggarwal and C. Zhai. *A Survey of Text Classification Algorithms*, pages 163–222. Springer US, Boston, MA, 2012.
3. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, Jan. 1991.
4. C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1):11–73, 1997.
5. R. Bouckaert. *Bayesian belief Networks: from construction to inference*. PhD thesis, 1995.
6. R. R. Bouckaert. *Bayesian network classifiers in Weka*. Department of Computer Science, University of Waikato Hamilton, 2007.
7. J. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. Brodley. Pruning decision trees with misclassification costs. *Machine Learning: ECML-98*, pages 131–136, 1998.
8. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
9. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001.
10. L. S. Cessie and J. C. van Houwelingen. Ridge Estimators in Logistic Regression. *Applied Statistics*, 41(1):191–201, 1992.
11. J. G. Cleary and L. E. Trigg. K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning*, pages 108–114. Morgan Kaufmann, 1995.
12. W. W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
13. G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.
14. K. Dembczyński, W. Cheng, and E. Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 279–287, 2010.
15. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.
16. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings 1995*, pages 194–202. Elsevier, 1995.
17. R.-E. Fan and C.-J. Lin. A study on threshold selection for multi-label classification. Technical report, 2007.
18. U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Thirteenth International Joint Conference on Articial Intelligence*, volume 2, pages 1022–1027. Morgan Kaufmann Publishers, 1993.
19. E. Frank and R. R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, pages 503–510, Berlin, Heidelberg, 2006. Springer-Verlag.
20. E. Frank, M. Hall, and B. Pfahringer. Locally weighted naive bayes. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 249–256, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

21. E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
22. Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, 1996. Morgan Kaufmann.
23. Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, Dec. 1999.
24. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2–3):131–163, Nov. 1997.
25. S. Godbole and S. Sarawagi. Discriminative methods for multi-labeled classification. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 22–30. Springer Berlin Heidelberg, 2004.
26. Y. Guo and S. Gu. Multi-label classification using conditional dependency networks. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1300–1305. AAAI Press, 2011.
27. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
28. T. Hastie and R. Tibshirani. Classification by pairwise coupling. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, NIPS '97, pages 507–513, Cambridge, MA, USA, 1998. MIT Press.
29. D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 85–96. AAAI Press, 1994.
30. A. S. Hesar, H. Tabatabaee, and M. Jalali. Structure learning of Bayesian networks using heuristic methods. In *Proceedings of International Conference on Information and Knowledge Management (ICIKM)*, 2012.
31. G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
32. T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
33. R. C. Holte. Very simple classification rules perform well on most commonly used datasets. 11(1):63–90, Apr. 1993.
34. T. Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms.* "page 40".
35. G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
36. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computing*, 13(3):637–649, Mar. 2001.
37. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.
38. J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 20(3), 1998.

39. R. Kohavi. The power of decision tables. In *Proceedings of the 8th European Conference on Machine Learning*, ECML '95, pages 174–189, London, UK, UK, 1995. Springer-Verlag.
40. L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017.
41. L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.
42. N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1):161–205, 2005.
43. F. H. Lars Kotthoff, Chris Thornton. User guide for auto-weka version 2.5. Technical report, Computer Science Department at University British Columbia (UBC), 2017.
44. D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European Conference on Machine Learning*, pages 4–15. Springer Berlin Heidelberg, 1998.
45. G. Madjarov, D. Kocev, D. Gjorgjevikj, and S. Dzeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*, 45(9):3084–3104, 2012.
46. A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *AAAI/ICML-98*, pages 41–48. AAAI Press, 1998.
47. J. C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.
48. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
49. J. Read. A pruned problem transformation method for multi-label classification. In *Proceedings of the New Zealand Computer Science Research Student Conference (NZCSRS)*, pages 143–150, 2008.
50. J. Read. *Scalable Multi-label Classification*. PhD thesis, 2010.
51. J. Read, L. Martino, and D. Luengo. Efficient monte carlo optimization for multi-label classifier chains. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP*, 2013.
52. J. Read, L. Martino, and D. Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3):1535–1546, Mar. 2014.
53. J. Read, L. Martino, P. M. Olmos, and D. Luengo. Scalable multi-output label prediction: From classifier chains to classifier trellises. *Pattern Recognition*, 48(6):2096–2109, 2015.
54. J. Read and F. Perez-Cruz. Deep learning for multi-label classification, 2014.
55. J. Read, B. Pfahringer, and G. Holmes. Multi-label classification using ensembles of pruned sets. In *IEEE International Conference on Data Mining*, pages 995–1000, Dec 2008.
56. J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, Dec. 2011.
57. J. Read, P. Reutemann, B. Pfahringer, and G. Holmes. MEKA: A multi-label/multi-target extension to Weka. *Journal of Machine Learning Research*, 17(21):1–5, 2016.
58. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
59. S. L. Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240, 1994.

60. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
61. M. Sumner, E. Frank, and M. Hall. Speeding up logistic model tree induction. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, pages 675–683, Berlin, Heidelberg, 2005. Springer-Verlag.
62. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of KDD-2013*, pages 847–855, 2013.
63. G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of the ECML/PKDD Workshop on Mining Multidimensional Data (MMD'08)*, pages 30–44, 2008.
64. G. Tsoumakas, I. Katakis, and I. Vlahavas. *Mining Multi-label Data*, pages 667–685. Springer US, Boston, MA, 2010.
65. G. Tsoumakas, I. Katakis, and I. Vlahavas. Random k-labelsets for multi-label classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, 2011.
66. B. Ustun, W. Melssen, and L. Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1):29 – 40, 2006.
67. J. Wicker, B. Pfahringer, and S. Kramer. Multi-label classification using boolean matrix decomposition. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC'12, pages 179–186, New York, NY, USA, 2012. ACM.
68. I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, third edition edition, January 2011.
69. D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.
70. J. H. Zaragoza, L. E. Sucar, E. F. Morales, C. Bielza, and P. Larrañaga. Bayesian chain classifiers for multidimensional classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 2192–2197. AAAI Press, 2011.
71. Z. Z. Zhang, M.L. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18:1338–1351, 2006.